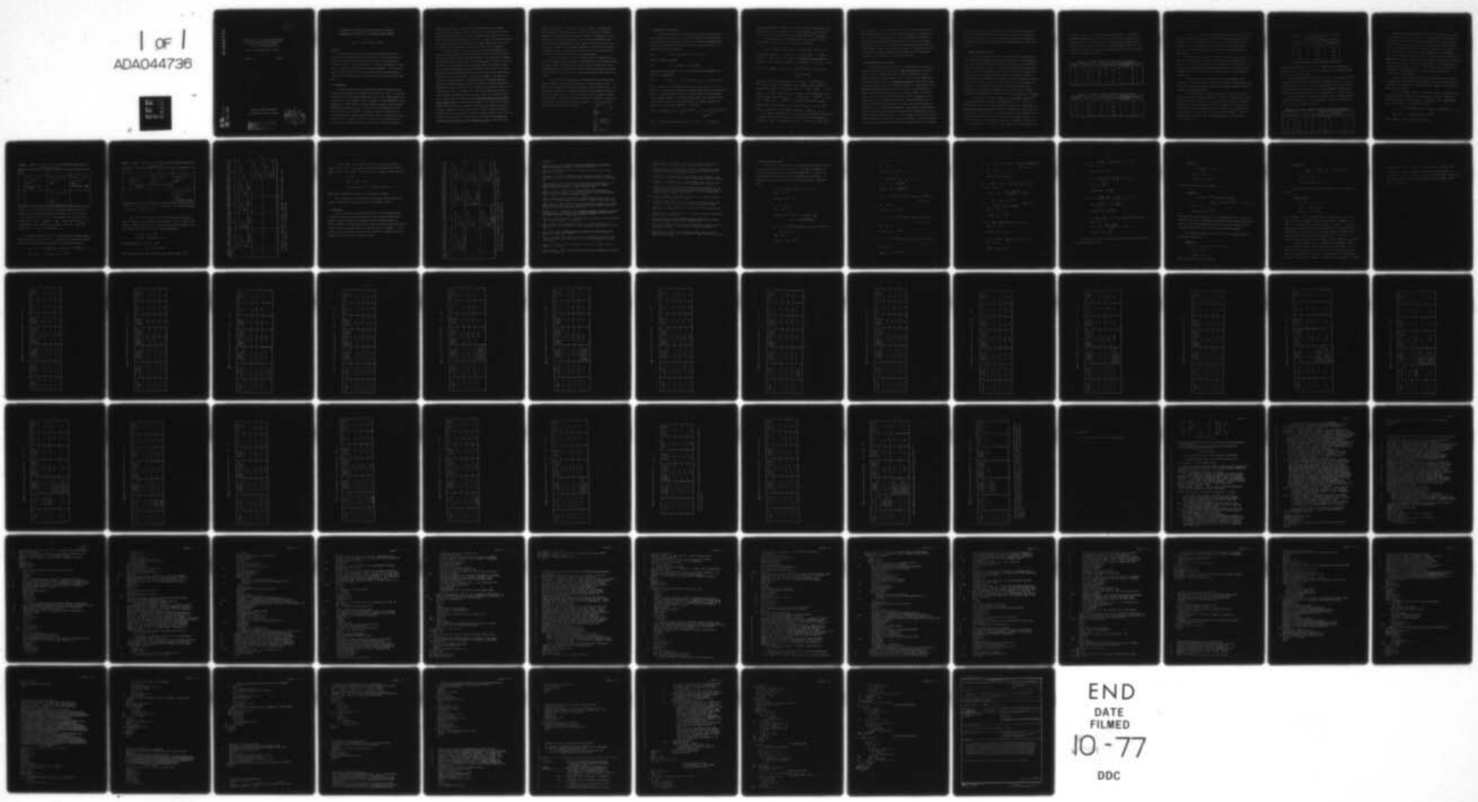
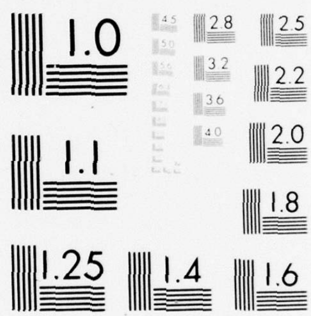
AD-A044 736 TEXAS UNIV AT AUSTIN CENTER FOR NUMERICAL ANALYSIS F/G 12/1
EXTRAPOLATION WITH SPLINE-COLLOCATION METHODS FOR TWO-POINT BOU--ETC(U)
AUG 77 J W DANIEL, A J MARTIN N00014-76-C-0275

UNCLASSIFIED CNA-125 NL

1 OF 1
ADA044736



END
DATE
FILMED
10-77
DDC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(12)

EXTRAPOLATION WITH SPLINE-COLLOCATION METHODS
FOR TWO-POINT BOUNDARY-VALUE PROBLEMS II:
C^2-CUBICS WITH DETAILED RESULTS

Andrew J. Martin and James W. Daniel

August 1977 CNA-125

CENTER FOR NUMERICAL ANALYSIS
THE UNIVERSITY OF TEXAS AT AUSTIN

Extrapolation with Spline-collocation Methods for Two-point Boundary-value Problems II: C^2-cubics with Detailed Results

by

Andrew J. Martin and James W. Daniel

Abstract

The methodology is very briefly described and then numerical results are presented for an implementation of a smooth-cubic-spline-collocation procedure accelerated via iterated deferred corrections to obtain approximate solutions, accurate to a prescribed tolerance, of two-point boundary-value problems for second-order scalar ordinary differential equations. The results are similar to those obtained via a less generally applicable finite-difference-oriented code described elsewhere which competes well with the best codes available.

1. Introduction

The primary methods for the approximate solution of nonlinear two-point boundary-value problems can be roughly classified into four types: (1) Rayleigh-Ritz-Galerkin and collocation methods [deBoor-Swartz (1973), Douglas-Dupont (1974), Russell (1974), Russell-Shampine (1972), *et cetera*], (2) shooting methods [Bulirsch *et alia* (1975), Keller (1968), Scott-Watts (1975), *et cetera*], and (4) standard finite-difference methods [Keller (1968, 1974, 1975), Lentini-Pereyra (1974, 1975a, 1975b), Pereyra (1968, 1973), *et cetera*]. Despite the dominance of the Rayleigh-Ritz-Galerkin and collocation methods in the more theoretical literature, there seems to be a consensus [Aziz (1975)] that the most efficient practical procedures are the shooting, imbedding, and difference methods, although to be

competitive the difference methods must be implemented with an acceleration technique such as extrapolation [Joyce (1971), Keller (1974, 1975)] or deferred correction [Fox (1947, 1962), Joyce (1971), Lentini-Pereyra (1974, 1975a, 1975b), Pereyra (1968, 1973), Daniel-Martin (1975, 1977)]. To make the collocation methods also competitive it has been proposed [Russell (1974), Daniel (1974)] that they be implemented with acceleration techniques much as are difference methods.

In [Daniel (1974)], the second of the present authors described how deferred correction could be used with various spline-collocation methods, although no results were given for any actual implementation. The present paper summarizes the numerical results obtained with a careful implementation of deferred correction applied to one of the spline-collocation methods proposed in that paper, namely the so-called extrapolated collocation method [Daniel-Swartz (1973), Hill (1973)] which uses twice-continuously-differentiable cubic spline functions and was motivated by the work of [Fyfe (1969)]. The code SPLIDC (spline iterated deferred correction) implementing this is a modification of the code NUMIDC (Numerov's method with iterated deferred correction) developed by the authors and shown in [Daniel-Martin (1975, 1977)] to be quite competitive with the best available codes on appropriate classes of problems. The present paper compares the behavior of SPLIDC with that of NUMIDC on the test problems earlier used for NUMIDC, namely a set of some seventeen second-order problems not involving the first derivative of the unknown function (so that Numerov's method will be fourth-order accurate); SPLIDC often follows the same computational paths as did NUMIDC and is about as reliable, although SPLIDC involves more effort in terms of function evaluations and much more total time because of overhead in solving systems of linear equations. We also present the results of using SPLIDC to solve some problems explicitly involving the first derivative in the differential equation, problems for which

Numerov's method drops to second-order accuracy (so NUMIDC is not applicable) while extrapolated collocation remains at fourth-order accuracy (so SPLIDC is applicable). Although SPLIDC was designed to handle the involvement of the first derivative in the boundary conditions, we have observed computationally some puzzling behavior and significantly greater numerical difficulties in this case; we therefore do <u>not</u> allow such boundary conditions in SPLIDC and we will devote considerable effort to resolving these difficulties as we proceed to study the use of different spline spaces (see the following paragraph).

Overall, our results serve as a feasibility study for using deferred correction with spline-collocation methods; they demonstrate that the methods proposed in [Daniel (1974)] can in fact work in practice. What is needed next is to determine what spaces of spline functions can be used most efficiently as the basis of such methods. We are pursuing such questions now and will report later on the results.

In the next section of this paper we describe briefly the method implemented by SPLIDC; for more detail the reader is referred to [Daniel (1974)]. In Section 3 we briefly describe how SPLIDC implements the method of Section 2; since the structure of SPLIDC is a slight modification of that of NUMIDC, the reader is referred to [Daniel-Martin (1975, 1977)] and to the code for SPLIDC in Section 8 for more detail. The numerical results are summarized in Section 4; a fairly complete report of the results can be found in Section 7. Section 5 presents our conclusions and Section 6 the references.

2. The Method to Be Implemented

A detailed description of iterated deferred correction used with extrapolated collocation may be found in Section 4 of [Daniel (1974)]; specifically, the material from the paragraph before Equation 4.6 through the paragraph following Equation 4.10 explains the process used. Here we will be very brief. We consider the differential equation

$$(2.1) \qquad y''(t) = f(t,y(t),y'(t)) \quad \text{for} \quad a \le t \le b$$

subject to boundary conditions

$$(2.2) \qquad y(a) = y_0 \text{ given}, \quad y(b) = y_F \text{ given},$$

and we consider approximating y by a twice-continuously-differentiable cubic spline Z on a uniform mesh

$$(2.3) \qquad a = t_0 < t_1 < \dots < t_N = b \quad \text{with } t_{i+1} - t_i = h = \frac{1}{N} \text{ for } 0 \le i \le N-1;$$

thus Z is given by a cubic polynomial on each piece (t_i, t_{i+1}) for $0 \le i \le N-1$. We let S be another cubic spline which interpolates the solution y to Equation 2.1 and 2.2 at the mesh points of Equation 2.3 and which satisfies some additional special boundary conditions. It follows [Daniel (1974)] that S satisfies differential equations similar to that in Equation 2.1; for example, writing g_i for $g(t_i)$, we have (for certain parameters e_j and d_j)

$$(2.4) \qquad S_i'' + \frac{1}{12}(S_{i-1}'' - 2S_i'' + S_{i+1}'') - \sum_{j=0}^{q-2} y_i^{(2j+6)} h^{2j+4} e_j = f(t_i, S_i, S_i' - \sum_{j=0}^{q-2} y_i^{(2j+5)} h^{2j+4} d_j) + \mathcal{O}(h^{2q+1})$$

for $i \le i \le N-1$, with similar equations for $i = 0$ and $i = N$. A sequence of

approximating splines Z is obtained essentially by using successively more terms (higher values of q) in Equation 2.4, yielding approximations accurate at the mesh points to successively higher orders: $\mathcal{O}(h^4)$, $\mathcal{O}(h^8)$, $\mathcal{O}(h^{12})$, <u>et cetera</u>.

More specifically, the zero-th <u>level</u> spline Z_0, with values $Z_{0,i}$ at the mesh points, satisfies $Z_{0,0} = y_0$ and $Z_{0,N} = y_F$ and solves

$$Z''_{0,i} + \frac{1}{12}(Z''_{0,i-1} - 2Z''_{0,i} + Z''_{0,i+1}) = f(t_i, Z_{0,i}, Z'_{0,i}) \quad \text{for } 1 \leq i \leq N-1$$

with similar equations at $i = 0$ and $i = N$; it turns out that $Z_0 = S + \mathcal{O}(h^4)$ so that $Z_0 = y + \mathcal{O}(h^4)$ and $Z_{0,i} = y_i + \mathcal{O}(h^4)$. The <u>deferred correction step</u> leads from Z_0 to Z_1, the solution at <u>level</u> one, via requiring $Z_{1,0} = y_0$, $Z_{1,N} = y_F$, and

$$(2.5) \quad Z''_{1,i} + \frac{1}{12}(Z''_{1,i-1} - 2Z''_{1,i} + Z''_{1,i+1}) - \sum_{j=0}^{0} Y_i^{(2j+6)} h^{2j+4} e_j = f(t_i, Z_{1,i}, Z'_{1,i} - \sum_{j=0}^{1} Y_i^{(2j+5)} h^{2j+4} d_j)$$

for $1 \leq i \leq N-1$, with similar equations for $i = 0$ and $i = N$; $Y_i^{(j)}$ denotes an h^{10-j}-accurate difference approximation to $y_i^{(j)}$ computed using the values $Z_{0,k}$ rather than the values y_k. It turns out that $Z_1 = S + \mathcal{O}(h^8)$ so that $Z_1 = y + \mathcal{O}(h^4)$ and $Z_{1,i} = y_i + \mathcal{O}(h^8)$. Similarly, further deferred correction steps lead to successively higher level solutions $Z_2, Z_3, \ldots$, satisfying $Z_m = S + \mathcal{O}(h^{4m+4})$, so that $Z_m = y + \mathcal{O}(h^4)$, $Z_{m,i} = y_i + \mathcal{O}(h^{4m+4})$. It is also possible easily to compute asymptotically correct estimates of the level-m errors $E_{m,i} = y_i - Z_{m,i}$ essentially by performing <u>one</u> step of Newton's method for finding $Z_{m+1,i} - Z_{m,i}$.

The simplest implementation of iterated deferred correction for extrapolated collocation, given a user-provided error tolerance TOL, would thus somehow select an initial $h = \frac{1}{N}$ and start computing $Z_0, E_0, Z_1, E_1, \ldots$ on this mesh, ideally until

Z_m (via E_m) meets the error criterion; if this does not succeed or at least progress satisfactorily, the code could increase N (for example, by doubling) and start over with the resulting new grid. This is essentially how Numerov's (or Cowell's) method was implemented in [Pereyra (1973)]. Since our code NUMIDC [Daniel-Martin (1975, 1977)] for that method incorporated many improvements in overall algorithm design, some of which were suggested in [Lentini-Pereyra (1974, 1975a, 1975b)], we likewise incorporated these improvements for extrapolated collocation in our code SPLIDC. We now turn to that code.

3. The Implementation of the Method

A detailed description of the practical modifications in the basic approach to iterated deferred correction for Numerov's method may be found in Sections 3 through 6 of [Daniel-Martin (1977)] and, in somewhat more detail, in Sections 3 through 7 of [Daniel-Martin (1975)]. Our code SPLIDC implementing iterated deferred correction for extrapolated collocation follows almost exactly that same structure as outlined for NUMIDC in Figure 7.1 of [Daniel-Martin (1977)]. Some slight modifications in structure were required in going from NUMIDC to SPLIDC to account for the differences in the two basic methods, the fact that SPLIDC allows y' to appear in the differential equation, *et cetera*. We still allow at most only 256 intervals in the mesh. A thoroughly commented code for SPLIDC may be found in Section 8 and is, of course, the ultimate authority on the structure of our algorithm's implementation. The fundamental improvements in SPLIDC, as in NUMIDC, and as compared with the simplest implementation sketched at the end of the preceding section, are: (1) avoiding useless deferred-correction steps at high levels by checking to see that the numerical solutions are behaving at the proper order asymptotically in h before increasing the level; (2) avoiding

unnecessary computations at refined meshes at low levels by using interpolated values from the high-level values on the coarser mesh to start at a non-zero level on the new mesh; (3) solving the nonlinear equations more efficiently and reliably; and (4) improving the control of both discretization errors and rounding errors.

4. Summary of Numerical Results

In this section we describe the overall results of our experience with SPLIDC and give some examples to illustrate specific features of the implementation. Data on the final accuracy and the cost to achieve it for each individual problem are presented in Section 7, so we merely summarize here; detailed information on the paths followed during the computation, the number of Newton iterations, et cetera, is available from the authors. All problems were solved by our code SPLIDC on the CDC 6600 at The University of Texas at Austin using the RUN compiler; the execution times given exclude output but are still somewhat indeterminate (to, say, within 5-10%) because timing involves a system request. In each case our codes attempt to approximate the true solution y by a spline Z within a given tolerance TOL in the sense that we desire $\max_{0\le i\le N} |y_i - Z_i| \le \mathrm{TOL} * \max_{0\le i\le N} |Z_i|$.

Our first table, Table 4.1, summarizes the performance of SPLIDC on thirteen non-pathological problems from [Daniel-Martin (1975)], all but three of which involve nonlinearities in y (polynomial, exponential, or logarithmic), but none of which explicitly involve y'. SPLIDC is provided with a logical switch so that the user can inform it if y' is absent from the equation, allowing considerable computational savings as should be obvious from Equations 2.4 and 2.5. We present for each of four tolerances the total time, total number of evaluations of f, and total number of evaluations of $\frac{\partial f}{\partial y}$, needed to solve all thirteen of the

problems; we also list, for each tolerance, the total number of occurrences of UNATTAINABLE ANNOUNCED (when the code announces that it cannot attain the requested tolerance) and of FAILURE (when the code incorrectly announces that it has attained the requested tolerance). This summary appears in Table 4.1; for comparison, we present the analogous data [Daniel-Martin (1977)] for the finite-difference code NUMIDC in Table 4.2.

Table 4.1 Performance Summary for SPLIDC on Thirteen Problems

TOLERANCE	TIME (SECS.)	f EVALUATIONS	$\frac{\partial f}{\partial y}$ EVALUATIONS	UNATTAINABLE ANNOUNCED	FAILURE
10^{-3}	1.403	1251	1183	0	0
10^{-6}	3.086	2963	2222	0	0
10^{-9}	5.927	5437	3167	0	0
10^{-12}	9.842	10341	5255	3	2

Table 4.2 Performance Summary for NUMIDC on Thirteen Problems

TOLERANCE	TIME (SECS.)	f EVALUATIONS	$\frac{\partial f}{\partial y}$ EVALUATIONS	UNATTAINABLE ANNOUNCED	FAILURE
10^{-3}	0.396	976	946	0	0
10^{-6}	1.156	2674	1866	0	0
10^{-9}	2.341	5264	2762	0	0
10^{-12}	4.947	10623	5244	2	0

From Table 4.1 we can see that iterated deferred correction for our spline-collocation method as implemented in SPLIDC does in fact work, that is, it does solve most of the test problems; this provides the practical justification for [Daniel (1974)]. By comparing Table 4.1 with Table 4.2, we also see, however, that in several respects SPLIDC does not perform as well on these problems as does NUMIDC:

(1) The two FAILUREs recorded for SPLIDC occurred because the actual error was four to five times the estimated error, which in turn barely met the tolerance; thus an accuracy of only about 3×10^{-12} rather than 10^{-12} was achieved. It should be noted that NUMIDC similarly underestimates the error on these thirteen problems about 10% of the time, but by coincidence this never caused FAILURE in the test cases.

(2) The much greater time required by SPLIDC than by NUMIDC is only slightly due to the greater number of function evaluations; it is primarily caused by the greater overhead involved in solving the more complex systems of equations involved in SPLIDC than in NUMIDC.

(3) The larger number of function evaluations is partially due to the desire to give this program the flexibility to handle y' in the differential equation or the boundary conditions. The current version reevaluates the function and derivative at the endpoints; while this is necessary if y' is present in the function or the boundary conditions, it is not necessary for the problems used to generate Tables 4.1 and 4.2 and could rather easily be eliminated. If these evaluations were eliminated the corresponding entries for SPLIDC would be as shown in Table 4.1a.

TABLE 4.1a Adjusted Performance Summary for SPLIDC on Thirteen Problems

Tolerance	f Evaluations	$\frac{\partial f}{\partial y}$ Evaluations
10^{-3}	1111	1051
10^{-6}	2727	2028
10^{-9}	5117	2939
10^{-12}	9909	4979

If we make this modification, then for the 47 cases in which both NUMIDC and SPLIDC reach the tolerance, SPLIDC requires no more evaluations than does NUMIDC in 31 cases, or 66% of the time.

Thus, it appears that if we want a deferred correction version of a spline collocation process to compete with NUMIDC for problems in which y' is not explicitly present, we must use a basic collocation process that is more efficient than is extrapolated collocation. We are presently examining the many spline-collocation methods available with the goal of finding such efficient processes.

Recall now that SPLIDC, as opposed to NUMIDC, is able to handle equations explicitly depending on y'. Table 4.3 summarizes the behavior of SPLIDC on a set of six such test problems in Section 7.

TABLE 4.3 Performance Summary for SPLIDC on Six Problems Involving y'

Tolerance	Time (secs.)	f Evaluations	$\frac{\partial f}{\partial y}$ or $\frac{\partial f}{\partial y'}$ Evaluations	Unattainable Announced	Failure
10^{-3}	0.673	693	627	0	1
10^{-6}	2.075	2431	1711	0	1
10^{-9}	3.954	5219	2421	1	0
10^{-12}	5.964	8957	3240	2	2

Again we see that iterated deferred correction of extrapolated collocation as implemented in SPLIDC does work to solve problems involving y' explicitly in the equation, thus providing the computational justification lacking in [Daniel (1974)]. The one FAILURE at TOL = 10^{-3} is relatively minor in that a tolerance of 2×10^{-3} was actually achieved; the FAILURE was caused by the fact that the true error was about two to three times the estimated error. The FAILURE at TOL = 10^{-6} was negligible since the tolerance actually obtained was 1.0018×10^{-6}, again caused by the fact that the true error was larger (by a factor of less than 1.33) than its estimate. One of the FAILUREs at TOL = 10^{-12} was slightly significant in that a tolerance of only 6.2×10^{-12} was achieved, due to true error being about 35 times estimated error; the other FAILURE was minor, since 1.3×10^{-12} was actually achieved.

In order to demonstrate that the deferred correction process is actually working and that SPLIDC does not simply remain at level zero with fourth-order accuracy from the basic method, we present now some examples of the computational paths followed by SPLIDC.

At a large tolerance like 10^{-3}, as in NUMIDC, usually the tolerance is met at level zero after changing the mesh spacing one or two times. On some more difficult problems, correction steps may be performed. This is exemplified for TOL = 10^{-3} in Problem #6 of [Daniel-Martin (1975)]:

$$y'' = y + y^3 + e^{\sin 4\pi x}\left[16\pi^2(\cos^2 4\pi x - \sin 4\pi x) - e^{2\sin 4\pi x} - 1\right] \text{ for } 0 < x < 1,$$

$$y(0) = y(1) = 1 \quad \text{with solution } y(x) = e^{\sin 4\pi x}.$$

The performance of SPLIDC is described in Table 4.4.

TABLE 4.4 PROBLEM #6 WITH TOL = 10^{-3}, SO THAT THE MAXIMUM ABSOLUTE ERROR MUST NOT EXCEED 2.7×10^{-3}

m = \ n =	8	16	32
0	1.5 @ 0 + 1.8 @ -5 (1.0 @ 0) NEWT = 5 No def. corr. step possible	8.8 @ -2 + 8.2 @ -6 (2.6 @ -2) NEWT = 4	1.2 @ -3 + 3.8 @ -4 (1.1 @ -3) NEWT = 2 (Convergence!) STOP
1		7.9 @ -1 + 3.3 @ -4 (1.1 @ -1) NEWT = 2 (Error increased.)	

An entry "a + b(c)" means that estimated absolute discretization error is a, a bound on the error in solving the nonlinear equations $G(z) = 0$ is b, while c is the actual total error. It is a+b that estimates the total error and is tested against 2.7×10^{-3} = YNORM x TOL. "NEWT = p" means that iterates $Z_0, \ldots, Z_p$ were generated to solve the nonlinear equations. Total time = .194, total f-evaluations = 170, total $\frac{\partial f}{\partial y}$ - evaluations = 137.

At a moderate tolerance like 10^{-6}, one correction step is quite common; on harder problems, more corrections occur. In Table 4.5 we illustrate the behavior of this code on a somewhat easier problem (#3 of [Daniel-Martin (1975)]):

$$y'' = y + y^3 + e^{\sin 2\pi x}\left[4\pi^2(\cos^2 2\pi x - \sin 2\pi x) - e^{2\sin 2\pi x} - 1\right] \text{ for } 0 < x < 1,$$

$$y(0) = y(1) = 1 \text{ with solution } y(x) = e^{\sin 2\pi x}.$$

TABLE 4.5 PROBLEM #3 WITH TOL = 10^{-6} SO THAT THE MAXIMUM ABSOLUTE ERROR MUST NOT EXCEED 2.7×10^{-6}

m \ n =	8	16	32
0	8.3 @ -2 + 2.8 @ -9 (2.0 @ -2) NEWT = 6 (No def. corr. step possible)	9.6 @ -4 + 2.9 @ -9 (1.1 @ -3) NEWT = 3 (Not at correct order)	6.4 @ -5 + 9.4 @ -7 (6.4 @ - 5) NEWT = 2
1			2.1 @ -7 + 4.5 @ -7 (7.1 @ -7) NEWT = 2 (Convergence!) STOP

See Table 4.4 for explanation of entries. Total time = .210, total f-evaluations = 202, total $\frac{\partial f}{\partial y}$ - evaluations = 169.

Tables 8.3 and 8.4 of [Daniel-Martin (1977)] give the (very similar) behavior of NUMIDC on these problems at these tolerances. We feel it is more interesting to consider problems with y' present for more difficult tolerances. The next example problem (#3YP of Section 7) is quite badly scaled.

$$y'' = 11y' + 12y - 22e^x$$
$$\text{with } y(0) = 1 \text{ and } y(15) = e^{15};$$

the exact solution is $y(x) = e^x$ so that

$$\|y\| = e^{15} \approx 3.26 \times 10^6.$$

Table 4.6 shows the path taken by SPLIDC on this problem with TOL = 10^{-9}.

TABLE 4.6 PROBLEM #3YP WITH TOL = 10^{-9}, SO THAT THE MAXIMUM ABSOLUTE ERROR MUST NOT EXCEED 3.26×10^{-3}

m = \ n =	8	16	32	64	128
0	5.6 @+4 + 1.6@-6 (4.7 @ 4) NEWT = 2 (No def. corr. step possible)	7.9 @ 3 + 5.2@-7 (6.9 @ 3) NEWT = 2 (Not at correct order)	2.2 @3 + 1.5@-7 (3.8 @ 2) NEWT = 2 (Not at correct order)	1.8 @1 + 3.9@-8 (1.9 @ 1) NEWT = 2 (Not at correct order)	1.3 @ 0 + 1.9 @ -8 (1.3 @ 0) NEWT = 2
1					8.9@-3 + 1.6@-8 (9.0@-3) NEWT = 2
2					5.6@-4 + 1.8@-8 (7.7 @ -5) NEWT = 2 (Convergence!) STOP

See Table 4.4 for explanation of entries. Total time = .789; total f-evaluations = 1022;
total $\frac{\partial f}{\partial y}$ - evaluations = 506 = total $\frac{\partial f}{\partial y'}$-evaluations.

The final example has a tolerance of 10^{-12}. As can be noted from the performance results, this is a very tight tolerance for this program; we will give an example in which it was attained. The problem we will consider is #5YP of Section 7:

$$y'' = 2yy'$$

$$y(0) = 0, \quad y(1) = \tan(1)$$

with exact solution $y = \tan x$, $\|y\| = \tan(1) \approx 1.5$.

The results for this problem with TOL $= 10^{-12}$ are given in Table 4.7.

This concludes our brief presentation of performance data for SPLIDC; more detailed results may be found in Section 7 of [Martin-Daniel (1977)].

5. Conclusions

The first conclusion is that SPLIDC does work. The test cases mentioned in Section 4 illustrate that iterated deferred corrections with collocation is a feasible solution technique for a substantial class of problems; however, the overall performance of SPLIDC leaves open the question of whether collocation methods with acceleration can be made competitive with other solution techniques. If collocation is to be made competitive with the other techniques, it seems clear that the appropriate spline space must be carefully chosen.

TABLE 4.7 PROBLEM #5YP WITH TOL = 10^{-12}, SO THAT THE MAXIMUM ABSOLUTE ERROR MUST NOT EXCEED 1.5 x 10^{-12}

m = \ n =	8	16	32	64
0	9.7 @ -5+1.3 @ -14 (8.8 @ -5) NEWT = 5 (No def. corr. step possible)	9.1 @ -6 + 3.0 @ -14 (9.1 @ -6) NEWT = 3	(6.9 @ -8) by interpolation from (2, 16)	(1.7 @ -8) by interpolation from (1, 32)
1		5.9 @ -8 + 3.1 @ -14 (7.1 @ -8) NEWT = 3	1.3 @ -9 + 3.8 @ -14 (1.3 @ -9) NEWT = 2 (Not at correct order)	1.6 @ -11 1.8 @ -13 (1.6 @ -11) NEWT = 2
2		1.0 @ -8 + 2.4 @ -13 (1.2 @ -8) NEWT = 2 (Error did not decrease enough)		4.3 @ -13 + 1.7 @ -13 (1.2 @ -12) NEWT = 2 (Convergence!) STOP

See Table 4.4 for explanation of entries. Total time = .527; total f-evaluations = 605;

total $\frac{\partial f}{\partial y}$ - evaluations = 194 = total $\frac{\partial f}{\partial y'}$ -evaluations.

6. References

1. Aziz, A. K. (ed.) (1975), Numerical Solution of Boundary-value Problems for Ordinary Differential Equations, Academic Press, New York.

2. deBoor, Carl, Blair Swartz (1973), "Collocation at Gaussian points," SIAM J. Num. Anal. 10, 582-606.

3. Bulirsch, R., J. Stoer, P. Deuflhard (1975), "Numerical solution of nonlinear two-point boundary-value problems," preprint of article to appear in Numerische Math.

4. Daniel, James W. (1974), "Extrapolation with spline-collocation methods for two-point-boundary-value problems I: Justifications and Proposals," CNA-89, Center for Numerical Analysis, UT Austin. Also Aeq. Math.

5. Daniel, J. W., A. J. Martin (1975), "Implementing deferred corrections for Numerov's difference method for second-order two-point boundary-value problems," CNA-107, Center for Numerical Analysis, UT Austin.

6. Daniel, James W., Andrew J. Martin (1977), "Numerov's method for deferred corrections for two-point boundary-value problems," to appear, SIAM J. Numer. Anal.

7. Daniel, James W., Blair K. Swartz (1973), "Extrapolated collocation for two-point boundary-value problems using cubic splines," Los Alamos Scientific Lab. Report No. LA-DC-72-1520; also JIMA (1975) 16, 161-174.

8. Douglas, Jim, Jr., Todd Dupont (1974), Collocation methods for parabolic equations in a single space variable based on C'-piecewise-polynomial, Springer Lecture Note Series, Vol. 385, Springer-Verlag, Berlin.

9. Fyfe, D. J. (1969), "The use of cubic splines in the solution of two-point boundary-value problems," Computer J. 12, 188-192.

10. Fox, L. (1947), "Some improvements in the use of relaxation methods for the solution of ordinary and partial differential equations," Proc. Royal Soc. London, A190, 31-59.

11. Fox, L. (ed.) (1962), Numerical Solution of Ordinary and Partial Differential Equations, Pergamon Press, Oxford.

12. Hill, T. R. (1973), "The solution of two-point boundary-value problems by extrapolated collocation with cubic splines," Univ. of Texas Mathematics Department M. A. thesis.

13. Joyce, C. C. (1971), "Survey of extrpolation processes in numerical analysis," SIAM Rev. 13, 435-490.

14. Keller, Herbert B. (1968), Numerical Methods for Two-point Boundary-value Problems, Blaisdell, Waltham, Mass.

15. Keller, Herbert B. (1974), "Accurate difference methods for nonlinear two-point boundary-value problems," SIAM J. Num. Anal. 11, 305-320.

16. Keller, Herbert B. (1975), "Numerical solution of boundary-value problems for ordinary differential equations: Survey and some recent results on difference methods," in [Aziz (1975)], 27-88.

17. Lentini,M., V. Pereyra (1974), "A variable-order finite-difference method for nonlinear multi-point boundary-value problems," Math. Comp. 28, 981-1004.

18. Lentini, M., V. Pereyra (1975a), "Boundary-problem solvers for first-order systems based on deferred corrections," in [Aziz (1975)], 293-316.

19. Lentini, M., V. Pereyra (1975b), "An adaptive finite-difference solver for nonlinear two-point boundary problems with mild boundary layers," manuscript, also presented at the NSF Regional Conference on Two-point Boundary-value Problems in Lubbock, Texas, in July 1975.

20. Pereyra, V. (1968), "Iterated deferred corrections for nonlinear boundary-value problems," Numer. Math. 11, 111-125.

21. Pereyra, V. (1973). "High-order finite difference solution of differential equations," Stanford Univ. Computer Sci. Dept. Report STAN-CS-73-348.

22. Russell, R. D. (1974), "Collocation for systems of boundary-value problems," Numer. Math. 23, 119-133.

23. Russell, R.D , L. F. Shampine (1972), "A collocation method for boundary-value problems," Numer. Math. 19, 1-28.

24. Scott, M. R. (1973), Invariant Imbedding and Its Applications to Ordinary Differential Equations, Addison-Wesley, Reading, Mass.

25. Scott, M. R. (1975), "On the conversion of boundary-value problems into stable initial-value problems via several invariant imbedding algorithms," in [Aziz (1975)], 89-148.

26. Scott, M. R., H. A. Watts (1975), "SUPORT--A computer code for two-point boundary-value problems via orthonormalization," Sandia Labs. Report 75-0198, Albuquerque, New Mexico.

7. Detailed Numerical Results

In this section we will list our test problems and the results we obtained for them using SPLIDC. We will list first, for the reader's convenience, the 17 test problems not involving y' from [Daniel-Martin (1975)]. The sources of these problems and some motivation for choosing them can be found there. The 13 problems used for Tables 4.1 and 4.2 are those below numbered 1 through 12 and 17.

1. $y'' = y^3 - (\sin x)(1 + \sin^2 x)$ for $0 < x < \pi$,

   $y(0) = y(\pi) = 0.$

   Solution: $y(x) = \sin x.$

2. $y'' = e^y$ for $0 < x < 1$,

   $y(0) = y(1) = 0.$

   Solution: $y(x) = -\ln 2 + 2 \ln \{c \sec \left[\frac{c}{2}(x - \frac{1}{2})\right]\}$

   where $c \approx 1.336055694906108....$

3. $y'' = y + y^3 + e^{\sin 2\pi x}[4\pi^2(\cos^2 2\pi x - \sin 2\pi x) - e^{2 \sin 2\pi x} - 1]$

   for $0 < x < 1$,

   $y(0) = y(1) = 1.$

   Solution: $y(x) = e^{\sin 2\pi x}.$

4. $y'' = \frac{1}{2}(y+x+1)^3$ for $0 < x < 1$,

$y(0) = y(1) = 0$.

Solution: $y(x) = 2(2-x)^{-1} - x - 1$.

5. $y'' = -.0003y/(.0001 + x^2)^2$ for $-.1 < x < .1$,

$-y(-.1) = y(.1) = .1/\sqrt{.0101}$

Solution: $y(x) = x/\sqrt{.0001 + x^2}$

6. $y'' = y + y^3 + e^{\sin 4\pi x}[16\pi^2(\cos^2 4\pi x - \sin 4\pi x) - e^{2\sin 4\pi x} - 1]$

for $0 < x < 1$,

$y(0) = y(1) = 1$.

Solution: $y(x) = e^{\sin 4\pi x}$

7. $y'' = y + y^5 + e^{\sin 2\pi x}[4\pi^2(\cos^2 2\pi x - \sin 2\pi x) - e^{4\sin 2\pi x} - 1]$,

for $0 < x < 1$,

$y(0) = y(1) = 1$

Solution: $y(x) = e^{\sin 2\pi x}$.

8. $y'' = y + y^5 + e^{\sin 4\pi x}[16\pi^2(\cos^2 4\pi x - \sin 4\pi x) - e^{4\sin 4\pi x} - 1]$

for $0 < x < 1$,

$y(0) = y(1) = 1$.

Solution: $y(x) = e^{\sin 4\pi x}$.

9. $y'' = y + y^3 + e^x\ \{4\pi\ \cos 2\pi x + \sin 2\pi x\,[-4\pi^2 - (\sin^2 2\pi x)\,e^{2x}]\}$

for $0 < x < 1$,

$y(0) = y(1) = 0$.

Solution: $y(x) = e^x \sin 2\pi x$.

10. $y'' = 400(y + \cos^2 \pi x) + 2\pi^2 \cos 2\pi x$ for $0 < x < 1$,

$y(0) = y(1) = 0$

Solution: $y(x) = \dfrac{e^{-20x} + e^{-20(1-x)}}{1+e^{-20}} - \cos^2 \pi x$.

11. $y'' = \begin{cases} 0 & \text{if } x \leq 0 \text{ or } y \leq 0 \\ 4x^{-6}y - 6y\ \ln^2 y & \text{otherwise} \end{cases}$ for $0 < x < 1$,

$y(0) = 0$, $y(1) = e^{-1}$.

Solution: $y(x) = e^{-1/x^2}$.

12. $y'' = 1600y - (1600+\pi^2)\sin \pi x$ for $0 < x < 1$,

$y(0) = y(1) = 0$.

Solution: $y(x) = \sin \pi x$.

13. $y'' = -(\pi^2 + .0001)y + .0001 \sin \pi x$ for $0 < x < 1$,

$y(0) = y(1) = 0$.

Solution: $y(x) = \sin \pi x$.

14. $y'' = -(\pi^2 + .00001)y + .00001 \sin \pi x$ for $0 < x < 1$,

$y(0) = y(1) = 0$.

Solution: $y(x) = \sin \pi x$.

15. $y'' = y^3 + \frac{40}{9}\left(x - \frac{1}{2}\right)^{2/3} - \left(x - \frac{1}{2}\right)^8$ for $0 < x < 1$,

$y(0) = y(1) = \left(\frac{1}{2}\right)^{8/3}$.

Solution: $y(x) = \left(x - \frac{1}{2}\right)^{8/3}$.

16. $y'' = y^3 + \frac{238}{9}\left(x - \frac{1}{2}\right)^{11/3} - \left(x - \frac{1}{2}\right)^{17}$ for $0 < x < 1$,

$y(0) = \left(-\frac{1}{2}\right)^{17/3}$, $y(1) = \left(\frac{1}{2}\right)^{17/3}$.

Solution: $y(x) = \left(x - \frac{1}{2}\right)^{17/3}$.

17. $y'' = 400(y + \cos^2 \pi x) + 2\pi^2 \cos 2\pi x + 400(e^y - e^{y^*(x)})$ for $0 < x < 1$,

$y(0) = y(1) = 0$,

where $y^*(x) = \dfrac{e^{-20x} + e^{-20(1-x)}}{1+e^{-20}} - \cos^2 \pi x$.

Solution: $y(x) = y^*(x)$.

We will now discuss in more detail the test problems we used that do involve y' in the differential equation.

Problem #1YP

$$y'' = \frac{-4xy'+2y}{1+x^2} \quad \text{for } 0 < x < 2,$$

$$y(0) = 1, \quad y(2) = .2$$

$$\text{Solution:} \quad y = 1/(1+x^2)$$

This problem was taken from [Fyfe (1969)].

Problem #2YP

$$y'' = y + \frac{1}{10}(y')^3 + e^{\sin 2\pi x}\{4\pi^2[\cos^2 2\pi x - \sin 2\pi x] - \frac{1}{10}\,8\pi^3\,(\cos 2\pi x)^3\, e^{2\sin 2\pi x} - 1\} \quad \text{for } 0 < x < 1,$$

$$y(0) = y(1) = 1.$$

$$\text{Solution:} \quad y(x) = e^{\sin 2\pi x}.$$

This problem was constructed from #3 by first substituting y' for y in the non-linearity. Some form of continuation method would be needed to handle that problem directly; to avoid complications not relevant to the present work, we used the factor $\frac{1}{10}$ so that the present algorithm can solve the initial system.

The remaining problems are from [Scott (1975)]. They have a variety of interesting characteristics.

Problem #3YP

$$y'' = 11y' + 12y - 22e^x \quad \text{for } 0 < x < 15,$$

$$y(0) = 1, \; y(15) = e^{15}.$$

$$\text{Solution:} \quad y(x) = e^x.$$

This linear problem is very badly scaled.

Problem #4YP

$$y'' = -\left[\frac{3}{\tan x} + 2\cdot\tan x\right] y' - \frac{7}{10} y \qquad \text{for } \frac{\pi}{6} < x < \frac{\pi}{3},$$

$$y\left(\frac{\pi}{6}\right) = 0, \quad y\left(\frac{\pi}{3}\right) = 5.$$

No exact solution known.

The final three problems are successively harder versions of one basic problem.

Problems #5YP-7YP

$$y'' = 2yy' \qquad \text{for } 0 < x < a < \frac{\pi}{2},$$

$$y(0) = 0, \quad y(a) = \tan(a).$$

Solution $y(x) = \tan(x)$.

As a approaches $\pi/2$, this problem becomes extremely difficult, since we are trying to follow the function's behavior as it approaches a singularity. For #5YP $a = 1$ and the problem is relatively simple. For #6YP $a = 1.5$ and the computation of an initial solution is quite difficult. To discover how the code would behave if it got started, we used the exact solution of the differential equation at the initial grid points as an initial guess. For #7YP $a = 1.55$ and this technique failed to produce an adequate initial solution for any tolerance.

For these problems, we will now summarize the performance of SPLIDC at each tolerance tested. For each problem and each tolerance (TOL = 10^{-3}, 10^{-6}, 10^{-9}, 10^{-12}) we list in Table 7.1 through 7.24 the actual errors achieved and the cost to achieve them. (Recall that a requested tolerance of TOL means that the goal is to make the error between the computed solution Y and the true solution

y^* satisfy $\|Y-y^*\|_\infty / \|y^*\|_\infty < TOL$.) As usual, an entry a @ b denotes $a \times 10^b$. If SPLIDC detects that the requested error cannot be attained, we will list the condition detected as well as the error we were able to attain. (The listed function counts include the unnecessary evaluation at the endpoints, as discussed in Section 4.)

TABLE 7.1 Performance Data for Problem #1. $\|y^*\|_\infty = 1.0$

Requested TOL	Actual $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Computed Bound on $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Final State (m,n) Where m = level and n = (b-a)/h	CDC 6600 Time in Seconds	Evaluations of f	Evaluations of $\frac{\partial f}{\partial y}$
1 @ - 3	1.7 @ - 5	1.8 @ - 5	(0,8)	.048	54	54
1 @ - 6	7.6 @ - 11	8.1 @ - 8	(1,16)	.112	105	88
1 @ - 9	9.4 @ - 11	9.5 @ - 11	(1,16)	.149	122	105
1 @ - 12	8.9 @ - 14	9.5 @ - 14	(2,16)	.171	165	114

TABLE 7.2 Performance Data for Problem #2. $\|y^*\|_\infty = .1137$

Requested TOL	Actual $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Computed Bound on $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Final State (m,n) Where m = level and n = (b-a)/h	CDC 6600 Time in Seconds	Evaluations of f	Evaluations of $\frac{\partial f}{\partial y}$
1 @ - 3	1.5 @ - 6	1.6 @ - 6	(0,8)	.029	27	27
1 @ - 6	1.3 @ - 7	1.3 @ - 7	(0,16)	.063	52	52
1 @ - 9	1.0 @ - 11	1.1 @ - 11	(1,16)	.083	78	61
1 @ - 12	1.5 @ - 13	7.4 @ - 14	(2,16)	.144	112	78

TABLE 7.3 Performance Data for Problem #3. $\|y^*\|_\infty = e \approx 2.718$

Requested TOL	Actual $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Computed Bound on $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Final State (m,n) Where m = level and n = (b-a)/h	CDC 6600 Time in Seconds	Evaluations of f	Evaluations of $\frac{\partial f}{\partial y}$
1 @ - 3	3.1 @ - 4	4.2 @ - 4	(0,16)	.074	61	61
1 @ - 6	2.5 @ - 7	7.2 @ - 8	(1,32)	.210	202	169
1 @ - 9	3.6 @ - 10	3.4 @ - 10	(1,64)	.410	358	227
1 @ - 12	2.21 @ -12 FAILURE!	4.8 @ - 13	(1,128)	.738	714	291

TABLE 7.4 Performance Data for Problem #4 $\|y^*\|_\infty = 3 - 2/2 \approx .172$

Requested TOL	Actual $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Computed Bound on $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Final State (m,n) Where m = level and n = (b-a)/h	CDC 6600 Time in Seconds	Evaluations of f	Evaluations of $\frac{\partial f}{\partial y}$
1 @ - 3	2.5 @ - 5	4.2 @ - 5	(0,8)	.032	27	27
1 @ - 6	1.2 @ - 8	1.2 @ - 8	(1,16)	.094	78	61
1 @ - 9	3.6 @ - 10	3.4 @ - 10	(2,16)	.141	112	78
1 @ - 12	3.5 @ - 13	1.5 @ - 13	(2,32)	.272	220	113

TABLE 7.5 Performance Data for Problem #5. $\|y^*\|_\infty = .995$

Requested TOL	Actual $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Computed Bound on $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Final State (m,n) Where m = level and n = (b-a)/h	CDC 6600 Time in Seconds	Evaluations of f	Evaluations of $\frac{\partial f}{\partial y}$
1 @ - 3	4.5 @ - 4	4.4 @ - 4	(0,64)	.231	189	189
1 @ - 6	3.4 @ - 8	5.9 @ - 8	(1,128)	.546	512	253
1 @ - 9	3.1 @ - 10	3.2 @ - 10	(1,256)	.935	897	381
1 @ - 12	3.0 @ - 5	Tolerance not attained, dominant rounding error (3.0 @ - 5)	(0,128)	.610	674	674

TABLE 7.6 Performance Data for Problem #6 $\|y^*\|_\infty = e \approx 2.718$

Requested TOL	Actual $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Computed Bound on $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Final State (m,n) Where m = level and n = (b-a)/h	CDC 6600 Time in Seconds	Evaluations of f	Evaluations of $\frac{\partial f}{\partial y}$
1 @ - 3	4.2 @ - 4	5.9 @ - 4	(0,32)	.194	170	153
1 @ - 6	2.6 @ - 7	7.8 @ - 8	(1,64)	.477	490	374
1 @ - 9	3.3 @ - 10	3.1 @ - 10	(1,128)	.791	798	471
1 @ - 12	3.3 @ - 7	Tolerance not attained, dominant rounding error (4.8 @ - 7)	(2,64)	.682	874	498

TABLE 7.7 Performance Data for Problem #7. $\|y^*\|_\infty = e \approx 2.718$

Requested TOL	Actual $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Computed Bound on $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Final State (m,n) Where m = level and n = (b-a)/h	CDC 6600 Time in Seconds	Evaluations of f	Evaluations of $\frac{\partial f}{\partial y}$
1 @ - 3	1.5 @ - 4	1.5 @ - 4	(0,16)	.137	114	114
1 @ - 6	2.3 @ - 7	6.6 @ - 8	(1,32)	.230	238	205
1 @ - 9	3.4 @ - 10	3.2 @ - 10	(1,64)	.439	385	254
1 @ - 12	9.2 @ - 13	3.7 @ - 13	(1,128)	.789	816	360

TABLE 7.8 Performance Data for Problem #8. $\|y^*\|_\infty = e \approx 2.718$

Requested TOL	Actual $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Computed Bound on $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Final State (m,n) Where m = level and n = (b-a)/h	CDC 6600 Time in Seconds	Evaluations of f	Evaluations of $\frac{\partial f}{\partial y}$
1 @ - 3	2.1 @ - 4	2.2 @ - 4	(0,32)	.243	256	222
1 @ - 6	2.4 @ - 7	7.1 @ - 8	(1, 64)	.501	543	410
1 @ - 9	3.2 @ - 10	3.0 @ - 10	(1,128)	.818	826	516
1 @ - 12	3.1 @ - 12 FAILURE!	6.6 @ - 13	(1,256)	1.576	1938	726

TABLE 7.9 Performance Data for Problem #9 $\|y^*\|_\infty \approx 2.144$

Requested TOL	Actual $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Computed Bound on $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Final State (m,n) Where m = level and n = (b-a)/h	CDC 6600 Time in Seconds	Evaluations of f	Evaluations of $\frac{\partial f}{\partial y}$
1 @ - 3	7.1 @ - 5	8.2 @ - 5	(0,16)	.062	61	61
1 @ - 6	7.7 @ - 8	8.0 @ - 8	(1,16)	.116	104	87
1 @ - 9	1.9 @ - 10	1.9 @ - 10	(1,32)	.238	187	103
1 @ - 12	1.1 @ - 13	1.5 @ - 13	(2,32)	.309	263	129

TABLE 7.10 Performance Data for Problem #10. $\|y^*\|_\infty \approx .774$

Requested TOL	Actual $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Computed Bound on $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Final State (m,n) Where m = level and n = (b-a)/h	CDC 6600 Time in Seconds	Evaluations of f	Evaluations of $\frac{\partial f}{\partial y}$
1 @ - 3	2.2 @ - 4	1.9 @ - 4	(1,16)	.080	60	43
1 @ - 6	5.5 @ - 8	5.4 @ - 8	(2,32)	.237	175	92
1 @ - 9	2.6 @ - 10	2.6 @ - 10	(2,64)	.485	305	124
1 @ - 12	1.4 @ - 13	8.5 @ - 14	(2,128)	.810	627	188

TABLE 7.11 Performance Data for Problem #11. $\|y^*\|_\infty = e^{-1} \approx .368$

Requested TOL	Actual $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Computed Bound on $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Final State (m,n) Where m = level and n = (b-a)/h	CDC 6600 Time in Seconds	Evaluations of f	Evaluations of $\frac{\partial f}{\partial y}$
1 @ - 3	1.1 @ - 5	3.4 @ - 5	(0,16)	.080	87	87
1 @ - 6	7.7 @ - 7	7.7 @ - 7	(0,32)	.234	253	253
1 @ - 9	2.8 @ - 10	2.5 @ - 11	(1,64)	.555	717	522
1 @ - 12	2.9 @ - 12	Tolerance not attained, cannot increase n. (5.9 @ - 12)	(1,256)	1.787	2325	1551

TABLE 7.12 Performance Data for Problem #12. $\|y^*\|_\infty = 1.$

Requested TOL	Actual $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Computed Bound on $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Final State (m,n) Where m = level and n = (b-a)/h	CDC 6600 Time in Seconds	Evaluations of f	Evaluations of $\frac{\partial f}{\partial y}$
1 @ - 3	4.1 @ - 7	4.1 @ - 7	(0,8)	.015	18	18
1 @ - 6	4.1 @ - 7	4.1 @ - 7	(0,8)	.016	18	18
1 @ - 9	3.0 @ - 10	4.7 @ - 10	(2,16)	.141	77	43
1 @ - 12	2.1 @ - 14	2.5 @ - 14	(1,64)	.379	265	100

TABLE 7.13 Performance Data for Problem #13. $\|y^*\|_\infty = 1.$

Requested TOL	Actual $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Computed Bound on $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Final State (m,n) Where m = level and n = (b-a)/h	CDC 6600 Time in Seconds	Evaluations of f	Evaluations of $\frac{\partial f}{\partial y}$
1 @ - 3	2.8 @ - 6	2.6 @ - 6	(1,64)	.302	319	189
1 @ - 6	1.8 @ - 7	2.0 @ - 8	(2,64)	.388	449	189
1 @ - 9	1.6 @ - 3	Tolerance not attained, dominant roundoff error. (1.6 @ - 3)	(0,64)	.384	428	428
1 @ - 12	3.1 @ - 6	Tolerance not attained, cannot increase n. (6.4 @ - 6)	(0,256)	1.400	1636	1636

TABLE 7.14 Performance Data for Problem #14 . $\|y^*\|_\infty = 1$

Requested TOL	Actual $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Computed Bound on $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Final State (m,n) Where m = level and n = (b-a)/h	CDC 6600 Time in Seconds	Evaluations of f	Evaluations of $\frac{\partial f}{\partial y}$
1 @ - 3	2.6 @ - 4	2.7 @ - 4	(1,64)	.317	319	189
1 @ - 6	7.7 @ - 6 FAILURE!	5.7 @ - 8	(1,128)	.689	805	351
1 @ - 9	3.1 @ - 5	Tolerance not attained, cannot increase n. (6.2 @ - 5)	(0,256)	1.430	1700	1700
1 @ - 12	2.9 @ - 5	Tolerance not attained, cannot increase n. (6.6 @ - 5)	(0,256)	1.446	1702	1702

TABLE 7.15 Performance Data for Problem #15. $\|y^*\|_\infty \approx .157$

Requested TOL	Actual $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Computed Bound on $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Final State (m,n) Where m = level and n = (b-a)/h	CDC 6600 Time in Seconds	Evaluations of f	Evaluations of $\frac{\partial f}{\partial y}$
1 @ - 3	2.1 @ - 2 FAILURE!	8.7 @ - 5	(0,16)	.077	52	52
1 @ - 6	6.7 @ - 4 FAILURE!	3.4 @ - 7	(0,128)	.522	408	408
1 @ - 9	2.1 @ - 4	Tolerance not attained, cannot increase n. (5.3 @ - 8)	(0,256)	1.158	1029	1029
1 @ - 12	2.1 @ - 4	Tolerance not attained, cannot increase n. (5.3 @ - 8)	(0,256)	1.403	1415	1415

TABLE 7.16 Performance Data for Problem #16. $\|y^*\|_\infty \approx .0197$.

Requested TOL	Actual $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Computed Bound on $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Final State (m,n) Where m = level and n = (b-a)/h	CDC 6600 Time in Seconds	Evaluations of f	Evaluations of $\frac{\partial f}{\partial y}$
1 @ - 3	3.3 @ - 4	5.9 @ - 4	(0,8)	.031	18	18
1 @ - 6	1.8 @ - 8	1.9 @ - 10	(1,32)	.204	151	101
1 @ - 9	1.8 @ - 8 FAILURE!	1.9 @ - 10	(1,32)	.206	168	118
1 @ - 12	7.2 @ - 12 FAILURE!	1.1 @ - 13	(1,128)	.567	458	214

TABLE 7.17 Performance Data for Problem #17 $\|y^*\|_\infty = .774$

Requested TOL	Actual $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Computed Bound on $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Final State (m,n) Where m = level and n = (b-a)/h	CDC 6600 Time in Seconds	Evaluations of f	Evaluations of $\frac{\partial f}{\partial y}$
1 @ - 3	5.8 @ - 5	1.1 @ - 4	(0,32)	.178	127	127
1 @ - 6	7.0 @ - 7	6.4 @ - 7	(1,32)	.250	193	160
1 @ - 9	6.4 @ - 10	6.4 @ - 10	(1,128)	.742	575	282
1 @ - 12	3.2 @ - 13	9.3 @ - 14	(2,256)	1.575	1348	443

TABLE 7.18 Performance Data for Problem #1YP $\|y^*\|_\infty = 1$

Requested TOL	Actual $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Computed Bound on $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Final State (m,n) Where m = level and n = (b-a)/h	CDC 6600 Time in Seconds	Evaluations of f	Evaluations of $\frac{\partial f}{\partial y}$ or $\frac{\partial f}{\partial y'}$
1 @ - 3	1.9 @ - 4	9.2 @ - 5	(0,8)	.013	18	18
1 @ - 6	2.8 @ - 9	2.6 @ - 9	(1,32)	.186	184	118
1 @ - 9	1.2 @ - 10	1.8 @ - 10	(2,32)	.254	250	118
1 @ - 12	1.2 @ - 12 FAILURE!	1.8 @ - 13	(2,64)	.528	640	183

TABLE 7.19 Performance Data for Problem #2YP $\|y^*\|_\infty = e \approx 2.718$

Requested TOL	Actual $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Computed Bound on $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Final State (m,n) Where m = level and n = (b-a)/h	CDC 6600 Time in Seconds	Evaluations of f	Evaluations of $\frac{\partial f}{\partial y}$ or $\frac{\partial f}{\partial y'}$
1 @ - 3	6.0 @ - 6	1.6 @ - 5	(0,32)	.176	170	170
1 @ - 6	6.9 @ - 7	2.7 @ - 7	(1,32)	.263	269	203
1 @ - 9	7.0 @ - 12	8.4 @ - 12	(1,128)	.868	1065	447
1 @ - 12	6.1 @ - 12 FAILURE!	2.0 @ - 13	(1,256)	1.533	2288	704

TABLE 7.20 Performance Data for Problem #3YP. $\|y^*\|_\infty = e^{15} \approx 3.26 \times 10^6$

Requested TOL	Actual $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Computed Bound on $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Final State (m,n) Where m = level and n = (b-a)/h	CDC 6600 Time in Seconds	Evaluations of f	Evaluations of $\frac{\partial f}{\partial y}$ or $\frac{\partial f}{\partial y'}$
1 @ - 3	1.2 @ - 4	6.7 @ - 4	(0,32)	.141	118	118
1 @ - 6	4.0 @ - 7	4.0 @ - 7	(0,128)	.512	506	506
1 @ - 9	2.3 @ - 11	1.7 @ - 10	(2,128)	.789	1022	506
1 @ - 12	1.6 @ - 12	Tolerance not attained, cannot increase n. (1.5 @ - 12)	(1,256)	1.236	1793	763

TABLE 7.21 Performance Data for Problem #4YP. $\|y^*\|_\infty = 5?$

Requested TOL	Computed Bound on $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Final State (m,n) Where m = level and n = (b-a)/h	CDC 6600 Time in Seconds	Evaluations of f	Evaluations of $\frac{\partial f}{\partial y}$ or $\frac{\partial f}{\partial y'}$
1 @ - 3	9.3 @ - 5	(0,8)	.031	18	18
1 @ - 6	1.3 @ - 8	(1,16)	.103	86	52
1 @ - 9	1.4 @ - 10	(1,32)	.310	219	85
1 @ - 12	7.9 @ - 13	(1,64)	.543	605	194

(The form of this table is different than the others since no exact solution is known. The computed norm is 5.)

TABLE 7.22 Performance Data for Problem #5YP. $\|y^*\| = \tan 1 \approx 1.56$

Requested TOL	Actual $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Computed Bound on $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Final State (m,n) Where m = level and n = (b-a)/h	CDC 6600 Time in Seconds	Evaluations of f	Evaluations of $\frac{\partial f}{\partial y}$ or $\frac{\partial f}{\partial y'}$
1 @ - 3	5.7 @ - 5	6.6 @ - 5	(0,8)	.031	27	27
1 @ - 6	4.5 @ - 8	3.8 @ - 8	(1,16)	.107	104	70
1 @ - 9	8.0 @ - 10	8.3 @ - 10	(1,32)	.281	254	120
1 @ - 12	7.7 @ - 13	3.9 @ - 13	(2,64)	.527	605	194

TABLE 7.23 Performance Data for Problem #6YP. $\|y^*\|_\infty = \tan 15 \approx 14.1$

Requested TOL	Actual $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Computed Bound on $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Final State (m,n) Where m = level and n = (b-a)/h	CDC 6600 Time in Seconds	Evaluations of f	Evaluations of $\frac{\partial f}{\partial y}$ or $\frac{\partial f}{\partial y'}$
1 @ - 3	2.2 @ - 3 FAILURE!	9.1 @ - 4	(1,32)	.281	342	276
1 @ - 6	1.002 @ - 6 FAILURE!	7.6 @ - 7	(1,128)	.904	1282	762
1 @ - 9	3.5 - 8	Tolerance not attained, cannot increase n. (3.8 @ - 8)	(1,256)	1.452	2409	1148
1 @ - 12	3.5 @ - 8	Tolerance not attained, cannot increase n. (3.8 @ - 8)	(1,256)	1.597	3053	1246

(On this problem an initial guess was given to start the Newton iteration.)

TABLE 7.24 Performance Data for Problem #7YP. $\|y^*\|_\infty = \tan 1.55 \approx 48.1$

Requested TOL	Actual $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Computed Bound on $\|Y-y^*\|_\infty / \|y^*\|_\infty$	Final State (m,n) Where m = level and n = (b-a)/h	CDC 6600 Time in Seconds	Evaluations of f	Evaluations of $\frac{\partial f}{\partial y}$ or $\frac{\partial f}{\partial y'}$
1 @ - 3	5.5 @ 1	Tolerance not attained, see note below. (8.3 @ 3)	(0,32)	.350	422	422
1 @ - 6						
1 @ - 9						
1 @ - 12						

(This problem is clearly too difficult for the code; even with an accurate initial guess, the actual and estimated error in the numerical solution increased as n increased. Ordinarily the code would try another linear interpolation of the boundary data; when an initial guess is given; giving up seems more realistic.)

8. The Code SPLIDC

The detailed listing of our code SPLIDC follows.

```
     SSS     PPPP     L          III    DDDD      CC
    S   S    P   P    L           I      D   D   C   C
    S        P   P    L           I      D   D   C
     SSS     PPPP     L           I      D   D   C
        S    P        L           I      D   D   C
    S   S    P        L           I      D   D   C   C
     SSS     P        LLLLL      III    DDDD      CCC

                  13.43.07  29 JUL 77
```

```
      SUBROUTINE SPLIDC(F,DFY,DFYP,N,A,B,ALF0,BV0,G0,ALFN,BVN,GN,W,X,
     1  TOL,NOYP,YINT)
C        THIS PROGRAM IMPLEMENTS ITERATED DEFERRED CORRECTION
C     ON A SPLINE COLLOCATION METHOD FOR
C
C                       (D**2)Y=F(X,Y,YP),
C
C          WITH GENERAL SEPARABLE LINEAR BOUNDARY CONDITIONS.
C
C            ALF0*Y(A)+BV0*YP(A)=G0 , ALFN*Y(B)+BVN*YP(B)=GN
C
C     AS DESCRIBED IN A FORTHCOMING REPORT.
C
C          WHILE THE METHOD SUPPORTS GENERAL LINEAR BOUNDARY CONDITIONS
C     THE PRESENT IMPLEMENTATION DOES NOT SUPPORT BOUNDARY CONDITIONS
C     INVOLVING YP.
C     *********  EXTRANEOUS FEATURES  ***********
C          THIS CODE CONTAINS A NUMBER OF FEATURES ORIENTED TOWARD
C     RESEARCH WORK RATHER THAN PRODUCTION WORK, SUCH AS TRACE-TYPE
C     PRINTING AND ESTIMATION OF COMPUTATIONAL EFFORT. THE VARIABLE
C       IFLAG   IS DECLARED AS COMMON AND TRACE-TYPE PRINTING OCCURS
C     IF AND ONLY IF IFLAG=0. THE COMMON VARIABLES   KOUNT1 AND KOUNT2
C     COUNT THE NUMBER OF FUNCTION EVALUATIONS PERFORMED.
C     SPECIAL TRACE PRINTING IS DONE
C     IN THE ROUTINE   PRSUB  .  ALL OF THESE ITEMS AND REFERENCES
C     TO THEM CAN BE REMOVED IN A PRODUCTION CODE.
C
C     ****** DESCRIPTION OF SUBROUTINE PARAMETERS   ********
C
C     F :  AN EXTERNAL USER-PROVIDED FUNCTION (OF TYPE REAL)
C          WITH CALLING SEQUENCE F(X,Y,YP); THIS SHOULD COMPUTE
C          THE FUNCTION F DEFINING THE DIFFERENTIAL EQUATION
C          FOR SIMPLE VARIABLES X,Y,YP.
C     DFY :  AN EXTERNAL USER-PROVIDED FUNCTION (ALSO OF TYPE
C          REAL) WITH CALLING SEQUENCE DFY(X,Y,YP) WHICH COMPUTES
C          THE CORRESPONDING VALUE OF DF/DY.
C     DFYP : ANOTHER EXTERNAL USER-PROVIDED FUNCTION (ALSO OF TYPE
C         REAL) WITH CALLING SEQUENCE DFYP(X,Y,YP) WHICH COMPUTES
C         THE CORRESPONDING VALUES OF DF/DYP.
C         IF NOYP IS TRUE, DFYP IS NEVER CALLED.
C     N :  AN OUTPUT PARAMETER. THE FINAL APPROXIMATE SOLUTION IS
C          PROVIDED ON A UNIFORMLY SPACED MESH OF WIDTH H=(XF-X0)/N.
C          AS PRESENTLY WRITTEN THIS CODE IS LIMITED TO N LESS OR
C          EQUAL 256; THIS CAN BE CHANGED BY APPROPRIATE CHANGES
C          IN DIMENSION STATEMENTS AND IN THE VALUES OF NMAX(=256)
C          AND NMAXH(=128).
```

```
C         IF YINT IS TRUE, N IS ALSO AN INPUT PARAMETER.
C     A : LEFT END POINT OF INTEGRATION INTERVAL,PROVIDED BY USER.
C     B : RIGHT END POINT OF INTEGRATION INTERVAL, PROVIDED BY
C         USER; A = B IS AN INVALID INPUT.
C     ALF0,BV0,G0 - COEFFICIENTS OF BOUNDARY CONDITION EQUATION AT A.
C     ALFN,BVN,GN - COEFFICIENTS OF BOUNDARY CONDITION EQUATION AT B.
C            BV0 AND BVN BOTH ZERO REMOVE YP FROM THE BOUNDARY CONDITIONS.
C        IT IS THIS CASE WHICH IS FULLY IMPLEMENTED.  THE PRESENT THEORY
C        REQUIRES THAT THE LINEAR INTERPOLATION PROBLEM HAVE A UNIQUE
C        SOLUTION; IF THIS IS NOT TRUE, EXECUTION IS TERMINATED.
C     W :  AN OUTPUT ARRAY, CONTAINING THE COMPUTED APPROXIMATE
C          SOLUTION.  UPON EXIT, THE FIRST FOUR COLUMNS OF W CORRESPOND
C          TO THE ELEMENTS OF X.  THEIR ELEMENTS ARE APPROXIMATIONS
C          TO THE FUNCTION VALUE AND THE FIRST THREE DERIVATIVES OF THE
C          SPLINE SOUGHT BY THIS NUMERICAL METHOD.
C          IF YINT IS TRUE, W IS ALSO AN INPUT PARAMETER.
C     X :  AN OUTPUT ARRAY CONTAINING THE N + 1 POINTS, INCLUDING
C          A AND B, AT WHICH THE SOLUTION IS APPROXIMATED.  UPON
C          EXIT, SPLIDC WILL HAVE X(1) = A , X(N+1) = B, AND
C          X(I) = A + (I-1)*H , WHERE H = (B - A) / N
C     TOL : REQUESTED ACCURACY, PROVIDED BY USER. SPLIDC ATTEMPTS
C          TO COMPUTE AN APPROXIMATE SOLUTION Y WITH A MAXIMUM ERROR
C          LESS THAN TOL TIMES THE MAXIMUM COMPUTED VALUE OF Y, A
C          SORT OF SCALED MAX-NORM ERROR.  TOL IS ALSO AN OUTPUT
C          PARAMETER, CONTAINING ON EXIT FROM SPLIDC THE ESTIMATED
C          UPPER BOUND ON THE ERROR; THIS ESTIMATE IS THE SUM OF THE
C          ASYMPTOTIC ESTIMATE OF THE DISCRETIZATION ERROR AND THE
C          ESTIMATE OF THE ERROR IN SOLVING THE NON-LINEAR EQUATIONS.
C          AN INPUT VALUE OF LESS THAN 1.E-12 WOULD LEAD TO UNRELIABLE
C          PERFORMANCE ON THIS MACHINE, SO SUCH VALUES ARE REPLACED
C          BY 1.E-12 WITH AN APPROPRIATE WARNING MESSAGE TO THE USER.
C          THE NUMBER 1.E-12 IS APPROPRIATE FOR THE CURRENT
C          CONFIGURATION. IN GENERAL WE WOULD RECOMMEND A NUMBER
C          APPROXIMATELY EQUAL TO THE PRODUCT OF THE MAXIMUM NUMBER OF
C          POINTS AND THE PRECISION OF THE MACHINE.
C     NOYP : LOGICAL VARIABLE.  IF NOYP IS TRUE, YP IS NOT INVOLVED
C            IN EITHER THE DIFFERENTIAL EQUATION OR THE BOUNDARY
C            CONDITIONS, SO THE ROUTINE SPLNYP IS USED.  OTHERWISE,
C            THE ROUTINE SPLWYP IS USED.
C     YINT : IF YINT IS TRUE, THE USER SUPPLIES AN INITIAL GUESS
C            TO THE SOLUTION Y.  (OTHERWISE, THE CODE BEGINS WITH
C            A LINEAR INTERPOLATION OF THE BOUNDARY DATA.)  IF YINT
C            IS TRUE N IS MEANINGFUL ON INPUT. THE FIRST
C            N+1 ELEMENTS OF THE FIRST COLUMN OF W SHOULD CONTAIN
C            THE DESIRED INITIAL APPROXIMATION TO THE SOLUTION.
C
C            SINCE THE DIFFERENCE IN THE CODE WITH AND WITHOUT
C      YP IS SUBSTANTIAL, WE SIMPLY BRANCH TO THE APPROPRIATE
C      ROUTINE DEPENDING ON NOYP.
C
      EXTERNAL F,DFY,DFYP
      DIMENSION W(259,5),X(259)
      LOGICAL NOYP,YINT
      IF(NOYP)GOTO 20
      CALL SPLWYP(F,DFY,DFYP,N,A,B,ALF0,BV0,G0,ALFN,BVN,GN,W,X,
     1   TOL,YINT)
      RETURN
```

```
20 CALL SPLNYP(F,DFY,DFYP,N,A,B,ALFO,BVO,GO,ALFN,BVN,GN,W,X,
  1    TOL,YINT)
   RETURN
   END




   SUBROUTINE SPLNYP(F,DFY,DFYP,N,A,B,ALFO,BVO,GO,ALFN,BVN,GN,W,X,
  1    TOL,YINT)
C     WE ALLOCATE STORAGE AT THIS POINT.  W IS USED TO STORE THE
C     MATRIX OF THE LINEARIZED SYSTEM DURING EXECUTION.
C     FU,E0,AND E1 STORE THE COMPUTED VALUES OF F DF/DY AND DF/DYP, RESP.
C     THE ARRAY S IS PRIMARILY USED TO STORE COMPUTED APPROXIMATIONS
C     TO THE DISCRETIZATION ERROR; THE ARRAY RHS IS USED IN THE
C     SOLUTION OF THE NON-LINEAR SYSTEM: RHS CONTAINS THE COMPUTED
C     RESIDUAL WHICH IS OVERWRITTEN WITH THE CHANGE IN Y.
C     WK IS A WORK AREA FOR THE LINEAR EQUATION SOLVER.
C     RHSTOT CONTAINS THE SPLINE COEFFICIENTS FOR THE COMPUTED
C     SOLUTION.  Y,YP AND AY2P CONTAIN COMPUTED APPROXIMATIONS
C     TO Y,YP AND THE DIFFERENCED SECOND DERIVATIVE OF THE
C     COMP. SOLUTION RESPECTIVELY.
      DIMENSION W(259,5),X(259),RHS(259),E0(259),E1(259),WK(777)
      DIMENSION RHSTOT(259),Y(259),YP(259),AY2P(259),FU(259),S(259)
C     WE WILL USE THE ARRAYS ERRSAV, TEST1, AND TEST2 IN THE TEST
C     FOR CORRECT ORDER.  THE SOLUTION AT LEVEL K=0,1,2,3,4, OR
C     5 SHOULD BEHAVE AT ORDER 4,8,12,16,20, OR 24 RESP.  WHEN
C     THE MESH-SPACING IS HALVED THE ERROR ESTIMATE SHOULD DECREASE
C     BY FACTORS OF 16,256,4096,ET CETERA. THE ELEMENTS OF
C     TEST1 AND TEST2 CONSTITUTE (RATHER COARSE) LOWER AND
C     UPPER BOUNDS, RESPECTIVELY, ON WHAT IS AN ACCEPTABLE LEVEL
C     OF DECREASE, WHILE ERRSAV SAVES THE ERRORS FROM THE PREVIOUS
C     N TO ALLOW COMPUTATION OF THESE FACTORS AND ALSO TO HELP
C     DETERMINE THE LEVEL AT WHICH INTERPOLATED VALUES ARE
C     ACCEPTED WHEN N IS DOUBLED.
      DIMENSION ERRSAV(6),TEST1(6),TEST2(6)
      DATA TEST1/8.,64.,512.,4096.,3.E4,2.E5/
      DATA TEST2/32.,1024.,3.E4,1.E6,3.E7,1.E9/
C     WE WILL USE LOCAL LOGICAL VARIABLES:
C          DIVNEW (TRUE WHEN NEWTON≠S METHOD HAS DIVERGED)
C          NEWN (TRUE WHEN WE ARE COMPUTING AT A NEW VALUE OF N)
C          RERROR (TRUE WHEN WE HAVE DETECTED DOMINANT ROUNDING ERROR)
C          CONVOB (TRUE WHEN WE HAVE HAD CONVERGENCE OF NEWTON≠S METHOD
C                 AT LEAST ONE TIME.)
C          NOEVAL (TRUE WHEN F AND DFY HAVE ALREADY BEEN EVALUATED.)
      LOGICAL YINT,RERROR,DIVNEW,CONVOB,NEWN,NOEVAL
      COMMON /FLAG/ IFLAG
      COMMON /KOUNT/ KOUNT1,KOUNT2
      DATA NMAX /256/
      DATA NMAXH /128/
C     CHECK THE VALIDITY OF THE INPUT PARAMETERS
      IF(A .EQ. B)STOP26
C     9.99E-13 EQUALS 1.E-12 - EPS FOR OUR PURPOSES
      IF(TOL .LE. 9.99E-13)STOP30
      IF(.NOT. YINT)N=8
C          INITIALIZE SOME VARIABLES
```

```
C     THE INITIAL VALUE OF N IS STORED AS N0; WHEN N=N0 NO BACKGROUND
C     INFORMATION EXISTS.
C     MAXK IS THE HIGHEST PREVIOUS LEVEL OF DEFERRED CORRECTION THAT
C     HAS BEEN REACHED, WHILE K IS THE PRESENT LEVEL.
C     EPNEWT IS THE TOLERANCE FOR CONVERGENCE IN NEWTON-S METHOD.
    2 MAXK=0
      N0=N
      EPSNWT=TOL/2.
      RERROR=.FALSE.
      CONVOB=.FALSE.
      K=0
C        BEGINNING OF COMPUTATION WITH A NEW VALUE OF N.
    3    ERROLD=1.E20
         NP1=N+1
         NP2=N+2
         NP3=N+3
C        KMAX IS MAXIMUM ALLOWABLE NUMBER OF EXTRAPOLATIONS FOR A
C        GIVEN N. THERE ARE THREE LIMITS.  THERE MUST BE ENOUGH POINTS
C        TO COMPUTE THE EXPANSIONS; 6 IS DETERMINED BY THE SIZE OF THE
C        RELEVANT ARRAYS; MAXK+2 IS A RESTRICTION ON THE NUMBER OF
C        EXTRAPOLATIONS WITHOUT HISTORY.
         KMAX=(NP1-4)/4
         KMAX=MIN0(KMAX,6,MAXK+2)
         NEWN=.TRUE.
         E0(1)=ALF0
         E1(1)=BV0
         E0(NP3)=ALFN
         E1(NP3)=BVN
         H=(B-A)/N
C             WE START PREPARING TO USE NEWTON-S METHOD TO SOLVE THE
C        NON-LINEAR EQUATIONS APPROPRIATE TO THE SPLINE COLLOCATION METHOD
C        USED.  THE VARIOUS CORRECTION TERMS MUST BE INITIALIZED
C        APPROPRIATELY, TO ZERO AT LEVEL ZERO.
C        ITLIM AND ITLIM2 LIMIT THE NUMBER OF NEWTON ITERATIONS AS
C        DESCRIBED LATER.
         ITLIM=10
         ITLIM2=13
         DO 6 I=2,NP2
    6    F1(I)=0.
         X(1)=A
         X(NP1)=B
         DO 10 I=2,N
   10    X(I)=X(1)+(I-1)*H
         DO 12 I=1,NP3
   12    S(I)=0.
         IF(N .EQ. N0)GOTO 19
C        FILL IN Y VALUES AFTER DOUBLING N
         CALL GETVAL(N,H,RHSTOT,Y,YP,AY2P)
C        AFTER AN INTERPOLATION, WE USE THE PREVIOUSLY COMPUTED VALUES
C        OF F AND DF/DY.  THE NEXT FEW LINES HANDLE THAT.
         NHALF=N/2
         N1=NHALF+1
         N2=NHALF+2
         F0(NP1)=F0(N1)
         F0(NP2)=E0(N2)
         DO 14 I=1,NHALF
            N0I=N1-I
```

```
         NI=NPI-2*I
         FU(NI)=FU(NOI)
         NI=NI+1
         E0(NI)=E0(NOI+1)
         FU(NI)=F(X(NI),Y(NI),YP(NI))
   14    E0(NI+1)=-DFY(X(NI),Y(NI),YP(NI))
      KOUNT1=KOUNT1+NHALF
      KOUNT2=KOUNT2+NHALF
      IF(K .EQ. 0)GOTO 38
      CALL MATMAN(W,RHS,E0,E1,WK,N,H,1)
      IF(IFLAG .GT. 0)GOTO 1002
      CALL PRSUB(N,28,ERRNOR,X,RHSTOT)
 1002 CONTINUE
C     NEXT, EVALUATE THE NEW CORRECTION S FOR THE NON-LINEAR
C     EQUATIONS THAT MUST BE SOLVED AT THE NEW LEVEL K AFTER AN
C     INTERPOLATION FOR THE VALUES ACCEPTED AT LEVEL K-1.
C     WHEN NEWN IS TRUE , DFY IS EVALUATED DURING THE NEWTON
C     ITERATIONS .
      NEWN=.FALSE.
      ITLIM=3
      ITLIM2=5
      CALL SPLDCG(K,N,FU,S,IERROR)
      DO 18 I=1,K
   18 ERRSAV(I)=ERRSAV(I)*16.**(-I)
      GOTO 38
   19 IF( .NOT. YINT)CALL LINST(G0,ALF0,GN,ALFN,A,B,N,H,RHSTOT)
      IF(YINT)CALL GETST(W,RHSTOT,WK,N,H)
      IF(IFLAG .EQ. 0)CALL PRSUB(N,28,TWOPI,X,RHSTOT)
C          MAIN PORTION OF NEWTON→S METHOD
C     WE ENTER HERE WITH AN INITIAL GUESS FOR Y, EITHER THE LINEAR
C     INTERPOLATION OF THE BOUNDARY DATA OR AN IMPROVED ESTIMATE
C     FROM A LOWER LEVEL AT THIS N OR A HIGHER LEVEL AT A SMALLER N
C     OR A USER SUPPLIED INITIAL GUESS.
C     AT LEVEL K=0 WE ALLOW 11 ITERATIONS BEFORE TERMINATING THE
C     PROCEDURE. AT HIGHER LEVELS, ONLY 4 ITERATIONS ARE ALLOWED.
C     AT LEVEL K=0 NEWN IS TRUE AND WE EVALUATE DFY AT EACH
C     ITERATE.  IN MOST CASES F AND DFY ARE EVALUATED BEFORE ENTERING THE
C     LOOP AND ARE NOT REEVALUATED ON THE FIRST ITERATION; IF NO SYSTEM
C     IS SOLVED AT K=0 THIS IS THE ONLY TIME DFY IS EVALUATED.
C     NORMAL EXIT OF THE NEWTON ITERATION OCCURS WHEN THE NORM
C     (RNORM) OF THE CHANGE IN THE SOLUTION Y IS LESS THAN OR
C     EQUAL TO HALF THE USER→S TOLERANCE TIMES THE NORM OF Y.
C     ABNORMAL EXITS:
C       (1) THE NORM OF THE RESIDUAL INCREASES FROM STEP N TO
C            N + 1, WHERE N > 0.
C            OR
C       (2) THE NUMBER OF ITERATIONS EXCEEDS ITLIM AND CONVERGENCE
C            IS NOT EXPECTED BY ITLIM2.
C     IF THE CHANGE IN Y (RNORM) INCREASES BUT THE RESIDUAL DECREASES,
C     THIS IS TAKEN AS A SIGN THAT ROUNDING ERROR IS PREVENTING
C     CONVERGENCE TO THE TOLERANCE AND THE PROGRAM GIVES AN ERROR
C     MESSAGE AND TERMINATES.
   38    INDEX=0
         DIVNEW=.FALSE.
         NOEVAL=.TRUE.
         IF((N .EQ. NO) .AND. (K .EQ. 0))NOEVAL=.FALSE.
   40       CALL GETVAL(N,H,RHSTOT,Y,YP,AY2P)
```

```
      RHS(1)=0.0
      RHS(NP3)=0.0
      RESN=AMAX1(ABS(RHS(1)),ABS(RHS(NP3)))
      IF(NOEVAL)GOTO 65
      DO 60 I=1,NP1
      YPTEMP=0.0
      FU(I)=F(X(I),Y(I),YPTEMP)
      IF( .NOT. NEWN)GOTO 60
      E0(I+1)=-DFY(X(I),Y(I),YPTEMP)
60    CONTINUE
      KOUNT1=KOUNT1+NP1
      IF( .NOT. NEWN)GOTO 65
      KOUNT2=KOUNT2+NP1
65    NOEVAL=.FALSE.
      DO 70 I=1,NP1
         RHS(I+1)=S(I+1)+FU(I)-AY2P(I)
         IF(RESN .LT. ABS(RHS(I+1)))RESN=ABS(RHS(I+1))
70       CONTINUE
      IJOB=2
      IF(NEWN)IJOB=0
      CALL MATMAN(W,RHS,E0,E1,WK,N,H,IJOB)
      RNORM=0.
      RTNORM=0.
      DO 100 I=1,NP3
      RHSTOT(I)=RHSTOT(I)+RHS(I)
      IF(RNORM .LT. ABS(RHS(I)))RNORM=ABS(RHS(I))
100   IF(RTNORM .LT. ABS(RHSTOT(I)))RTNORM=ABS(RHSTOT(I))
C     AYNORM IS AN APPROXIMATION TO THE NORM OF Y BASED ON THE
C     THEORETICAL BOUND ON THE RATIO OF THE NORMS OF THE SPLINE COEF.
C     AND THE FUNCTION Y.
      AYNORM=RTNORM/3.
      INDEX=INDEX+1
      IF(INDEX .EQ. 1)RNORM1=RNORM
      EPS=EPSNWT*AYNORM
      IF(RNORM .LE. EPS)GOTO 390
      IF(INDEX .LE. 2) GOTO 120
      IF(RESN .GT. RSNSAV)GOTO 300
      IF((RNORM .GT. RNSAV) .AND. CONVOB)GOTO 350
120   RNS2=RNSAV
      RNSAV=RNORM
      RSNSAV=RESN
      IF(INDEX .LE. ITLIM)GOTO 40
      IF(INDEX .GT. ITLIM2)GOTO 300
      IF((RNORM *(RNORM/RNS2)**(ITLIM2-INDEX)) .LT. EPS)GOTO 40
C    NEWTON+S METHOD IS DIVERGING AND WE MUST DOUBLE N IF
C    POSSIBLE.  IF THE FINAL NEWTON ITERATE SOLVES THE NON-
C    LINEAR SYSTEM BETTER THAN THE FIRST ITERATE, THEN WE GET
C    AN ERROR ESTIMATE FOR USE IN TESTING ORDER AND WE USE
C    VALUES INTERPOLATED FROM THE LAST ITERATE IN ORDER TO
C    START NEWTON+S METHOD AT LEVEL ZERO AND WITH N DOUBLED:
C    IF THE FINAL ITERATE WAS WORSE, HOWEVER, WE START WITH
C    THE LINEAR INTERPOLATION OF THE BOUNDARY DATA.
300  DIVNEW=.TRUE.
     IF((RNORM1 .GT. RNORM) .OR. CONVOB)GOTO 400
     IF(N .GT. NMAXH)GOTO 800
     IF(YINT)GOTO 820
     N=2*N
```

```
      GOTO 2
C     WE NOW THINK THAT ROUNDING ERROR HAS CONTAMINATED OUR
C     RESULTS, SO WE PREPARE TO QUIT COMPLETELY AFTER ESTIMATING
C     THE ERROR WE DID ACHIEVE.
  350 RERROR=.TRUE.
      GOTO 400
C     WE NOTE FOR FUTURE REFERENCE THAT CONVERGENCE HAS BEEN
C     OBTAINED AT LEAST ONCE.
  390 CONVOB=.TRUE.
C     WE WANT TO ESTIMATE THE DISCRETIZATION ERROR, ESSENTIALLY
C     BY ONE STEP OF NEWTON'S METHOD.  WE SET UP THE RIGHT HAND
C     SIDE RHS NEEDED FOR THE LINEAR EQUATIONS OF THIS ONE STEP;
C     WHILE COMPUTING RHS WE WILL ALSO COMPUTE S, THE CORRECTION
C     (ESTIMATING LOCAL TRUNCATION ERROR) THAT WE WILL
C     NEED FOR THE NON-LINEAR EQUATIONS AT THE NEXT LEVEL UP IF
C     WE GET THAT FAR.
  400 K=K+1
      ITLIM=3
      ITLIM2=5
      CALL SPLDCG(K,N,FU,RHS,IERROR)
      RHS(1)=0.0
      RHS(NP3)=0.0
      DO 450 I=2,NP2
      STF=RHS(I)
      RHS(I)=S(I)-STF
  450 S(I)=STF
C     WITH THE NEW RIGHT HAND SIDE, WE GENERATE AND SOLVE THE
C     NEW SYSTEM.
      CALL MATMAN(W,RHS,E0,E1,WK,N,H,2)
C
C     ERROR CONTROL AND DECISION CENTER
C     RHS NOW CONTAINS OUR ESTIMATE OF DISCRETIZATION ERROR.
C     WE COMPUTE ITS NORM IN ERRNOR, ALLOWING US TO TEST FOR
C     CONVERGENCE OF THE OVERALL ALGORITHM.
      NEWN=.FALSE.
      ERRNOR=0.
      YNORM=0.
      DO 460 I=1,NP1
      TE=ABS((RHS(I)+4.*RHS(I+1)+RHS(I+2))/6.)
      IF(TE .GT. ERRNOR)ERRNOR=TE
  460 IF(YNORM .LT. ABS(Y(I)))YNORM=ABS(Y(I))
      IF(IFLAG .GT. 0)GOTO 1001
      K1=K-1
      PRINT 470,INDEX,RESN,RNORM
      CALL PRSUB(N,K1,ERRNOR,X,RHSTOT)
 1001 CONTINUE
C     NEXT WE TEST OUR ESTIMATE OF OVERALL ERROR AGAINST
C     THE USER'S TOLERANCE.  ERRNOR ESTIMATES THE
C     DISCRETIZATION ERROR FOR THE EXACT SOLUTION OF THE
C     NON-LINEAR SYSTEM, WHILE RNORM ESTIMATES THE ERROR IN
C     SOLVING THAT NON-LINEAR SYSTEM.  ERREST ESTIMATES THE
C     TOTAL ERROR UNDER THE PESSIMISTIC ASSUMPTION THAT
C     THE OTHER ERRORS SUM.
      ERREST=ERRNOR+RNORM
      TOLLOC=TOL*YNORM
C     TEST FOR SUCCESS
      IF(ERREST .LT. TOLLOC)GOTO 900
```

```
C             TERMINATE ALGORITHM IF RERROR TRUE
              IF(RERROR)GOTO 980
C             IF K .LE.  MAXK (SO THAT THERE IS HISTORY TO COMPARE
C             AGAINST) WE NEXT CHECK WHETHER THE ERROR IS BEHAVING
C             AT THE CORRECT ORDER.
              IF(K .GT. MAXK)GOTO 480
              RAT=ERRSAV(K)/ERRNOR
              ERRSAV(K)=ERRNOR
              IF(IFLAG .EQ. 0)PRINT 475,RAT
C             THIS IS THE TEST FOR CORRECT ORDER.
              IF(RAT .LT. TEST1(K) .OR. RAT .GT. TEST2(K))GOTO 700
  480         ERRSAV(K)=ERRNOR
C             THE NEXT STATEMENT TESTS WHETHER THE PREVIOUS DEFERRED
C             CORRECTION IMPROVED THE ACCURACY ACCEPTABLY AND
C             WHETHER AN ERROR ESTIMATE WOULD BE ALLOWED AFTER
C             THE NEXT CORRECTION.
              IF(ERRNOR .GE. .1*ERROLD .OR. K+1 .GT. KMAX)GOTO 700
C             CHECK FOR NEWTON DIVERGENCE
              IF(DIVNEW)GOTO 700
              ERROLD=ERRNOR
C             GO AHEAD AND SOLVE THE SYSTEM AT THE HIGHER ORDER
              GOTO 38
C         WE NEED TO INCREASE N AND THE NEXT SECTION OF CODE HANDLES
C         THAT.  WE FIRST COMPUTE THE LEVEL INTERPOLATED VALUES WILL BE
C         ACCEPTED AT. THEN WE DO THE INTERPOLATION.
  700     IF(N .GT. NMAXH)GOTO 800
          MAXK=K
          NOLD=N
          N=2*N
          KINT=K
          ERROLD=AMIN1(ERROLD,ERRNOR)
C         DECIDE LEVEL TO INTERPOLATE TO
          DO 710 I=1,K
          IF(ERROLD .LT. ERRSAV(I)*16.**(-I))GOTO  710
          GOTO 720
  710     CONTINUE
  720     K=I-1
          IF(DIVNEW)K=0
C         INSPL PERFORMS THE INTERPOLATION AND FILLS IN Y VALUES.
          CALL INSPL(KINT,N,Y,W,S,RHSTOT,WK,K,H)
          GOTO 3
C     THIS IS THE N TOO BIG EXIT
  800 PRINT 815
      GOTO 900
C     820 IS THE ERROR EXIT TAKEN WHEN AN INITIAL GUESS FAILS.
  820 PRINT 825
C     THE FOLLOWING CODE SEGMENT PREPARES TO RETURN TO THE USER.
C     IN THE ABSENCE OF ERROR MESSAGES. THE RETURN IS SUCCESSFUL.
  900 TOL=ERREST
      IF(IFLAG .EQ. 0)PRINT 920,YNORM
      CALL SPLGEN(W,N,H,RHSTOT)
      RETURN
C     980 IS THE ROUNDING ERROR EXIT.
  980 PRINT 990
      GOTO 900
  470 FORMAT(I10,2F20.10)
  475 FORMAT(80X,F12.4)
```

```
  815 FORMAT(* N TOO BIG*)
  825 FORMAT(* ILLEGAL TO REINITIALIZE WHEN INITIAL GUESS GIVEN*)
  920 FORMAT(* YNORM = *,E20.10)
  990 FORMAT(*  ROUNDING ERROR EXIT*)
      END
```

```
      SUBROUTINE SPLWYP(F,DFY,DFYP,N,A,B,ALF0,BV0,G0,ALFN,BVN,GN,W,X,
     1  TOL,YINT)
C     WE ALLOCATE STORAGE AT THIS POINT.  W IS USED TO STORE THE
C     MATRIX OF THE LINEARIZED SYSTEM DURING EXECUTION.
C     FU,E0,AND E1 STORE THE COMPUTED VALUES OF F DF/DY AND DF/DYP, RESP.
C     THE ARRAY S IS PRIMARILY USED TO STORE COMPUTED APPROXIMATIONS
C     TO THE DISCRETIZATION ERROR; THE ARRAY RHS IS USED IN THE
C     SOLUTION OF THE NON-LINEAR SYSTEM; RHS CONTAINS THE COMPUTED
C     RESIDUAL WHICH IS OVERWRITTEN WITH THE CHANGE IN Y.
C     WK IS A WORK AREA FOR THE LINEAR EQUATION SOLVER.
C     RHSTOT CONTAINS THE SPLINE COEFFICIENTS FOR THE COMPUTED
C     SOLUTION.  Y,YP AND AY2P CONTAIN COMPUTED APPROXIMATIONS
C     TO Y,YP AND THE DIFFERENCED SECOND DERIVATIVE OF THE
C     COMP. SOLUTION RESPECTIVELY.
      DIMENSION W(259,5),X(259),RHS(259),E0(259),E1(259),WK(777)
      DIMENSION RHSTOT(259),Y(259),YP(259),AY2P(259),FU(259),S(259)
C     YPC CONTAINS THE CURRENT VALUES OF THE YP CORRECTION.  YPCOLD
C     CONTAINS THE PREVIOUS VALUES.
      DIMENSION YPC(259),YPCOLD(259)
C     WE WILL USE THE ARRAYS ERRSAV, TEST1, AND TEST2 IN THE TEST
C     FOR CORRECT ORDER.  THE SOLUTION AT LEVEL K=0,1,2,3,4, OR
C     5 SHOULD BEHAVE AT ORDER 4,8,12,16,20, OR 24 RESP.  WHEN
C     THE MESH-SPACING IS HALVED THE ERROR ESTIMATE SHOULD DECREASE
C     BY FACTORS OF 16,256,4096,ET CETERA. THE ELEMENTS OF
C     TEST1 AND TEST2 CONSTITUTE (RATHER COARSE) LOWER AND
C     UPPER BOUNDS, RESPECTIVELY, ON WHAT IS AN ACCEPTABLE LEVEL
C     OF DECREASE, WHILE ERRSAV SAVES THE ERRORS FROM THE PREVIOUS
C     N TO ALLOW COMPUTATION OF THESE FACTORS AND ALSO TO HELP
C     DETERMINE THE LEVEL AT WHICH INTERPOLATED VALUES ARE
C     ACCEPTED WHEN N IS DOUBLED.
      DIMENSION ERRSAV(6),TEST1(6),TEST2(6)
      DATA TEST1/8.,64.,512.,4096.,3.E4,2.E5/
      DATA TEST2/32.,1024.,3.E4,1.E6,3.E7,1.E9/
C     WE WILL USE LOCAL LOGICAL VARIABLES:
C          DIVNEW (TRUE WHEN NEWTON≠S METHOD HAS DIVERGED)
C          NEWN (TRUE WHEN WE ARE COMPUTING AT A NEW VALUE OF N)
C          RERROR (TRUE WHEN WE HAVE DETECTED DOMINANT ROUNDING ERROR)
C          CONVOB (TRUE WHEN WE HAVE HAD CONVERGENCE OF NEWTON≠S METHOD
C                 AT LEAST ONE TIME.)
C          NOEVAL (TRUE WHEN F AND DFY HAVE ALREADY BEEN EVALUATED.)
      LOGICAL YINT,RERROR,DIVNEW,CONVOB,NEWN,NOEVAL
      COMMON /FLAG/ IFLAG
      COMMON /KOUNT/ KOUNT1,KOUNT2
      DATA NMAX /256/
      DATA NMAXH /128/
C     CHECK THE VALIDITY OF THE INPUT PARAMETERS
```

```
      IF(A .EQ. B)STOP26
C     FOR THE PRESENT, WE STOP WHEN THE LINEAR INTERPOLATION
C     PROBLEM HAS NO SOLUTION.
      IF(((ALFO*A+BVO)*ALFN) .EQ. ((ALFN*B+BVN)*ALFO))STOP27
C     9.99E-13 EQUALS 1.E-12 - EPS FOR OUR PURPOSES
      IF(TOL .LE. 9.99E-13)STOP30
      IF( .NOT. YINT)N=8
C           INITIALIZE SOME VARIABLES
C     THE INITIAL VALUE OF N IS STORED AS N0: WHEN N=N0 NO BACKGROUND
C     INFORMATION EXISTS.
C     MAXK IS THE HIGHEST PREVIOUS LEVEL OF DEFERRED CORRECTION THAT
C     HAS BEEN REACHED, WHILE K IS THE PRESENT LEVEL.
C     EPNEWT IS THE TOLERANCE FOR CONVERGENCE IN NEWTON+S METHOD.
    2 MAXK=0
      N0=N
      EPSNWT=TOL/2.
      RERROR=.FALSE.
      CONVOB=.FALSE.
      K=0
C     BEGINNING OF COMPUTATION WITH A NEW VALUE OF N.
    3    ERROLD=1.E20
         NP1=N+1
         NP2=N+2
         NP3=N+3
C        KMAX IS MAXIMUM ALLOWABLE NUMBER OF EXTRAPOLATIONS FOR A
C        GIVEN N. THERE ARE THREE LIMITS.  THERE MUST BE ENOUGH POINTS
C        TO COMPUTE THE EXPANSIONS; 6 IS DETERMINED BY THE SIZE OF THE
C        RELEVANT ARRAYS; MAXK+2 IS A RESTRICTION ON THE NUMBER OF
C        EXTRAPOLATIONS WITHOUT HISTORY.
         KMAX=(NP1-4)/4
         KMAX=MIN0(KMAX,6,MAXK+2)
         NEWN=.TRUE.
         E0(1)=ALFO
         E1(1)=BVO
         E0(NP3)=ALFN
         E1(NP3)=BVN
         H=(B-A)/N
C              WE START PREPARING TO USE NEWTON+S METHOD TO SOLVE THE
C        NON-LINEAR EQUATIONS APPROPRIATE TO THE SPLINE COLLOCATION METHOD
C        USED.  THE VARIOUS CORRECTION TERMS MUST BE INITIALIZED
C        APPROPRIATELY, TO ZERO AT LEVEL ZERO.
C        ITLIM AND ITLIM2 LIMIT THE NUMBER OF NEWTON ITERATIONS AS
C        DESCRIBED LATER.
         ITLIM=10
         ITLIM2=13
         X(1)=A
         X(NP1)=B
         DO 10 I=2,N
   10    X(I)=X(1)+(I-1)*H
         DO 12 I=1,NP3
         YPC(I)=0.
         YPCOLD(I)=0.
   12    S(I)=0.
         IF(N .EQ. N0)GOTO 19
C        FILL IN Y VALUES AFTER DOUBLING N
         CALL GETVAL(N,H,RHSTOT,Y,YP,AY2P)
C              WE USE DIFFERENT YP VALUES AFTER AN INTERPOLATION
```

```
C           TO EVALUATE THE FUNCTION, SO EVALUATION IS REPEATED.
            DO 15 I=1,NP1
            FU(I)=F(X(I),Y(I),YP(I))
            E0(I+1)=-DFY(X(I),Y(I),YP(I))
   15       E1(I+1)=-DFYP(X(I),Y(I),YP(I))
            KOUNT1=KOUNT1+NP1
            KOUNT2=KOUNT2+NP1
            IF(K .EQ. 0)GOTO 38
            CALL MATMAN(W,RHS,E0,E1,WK,N,H,1)
            IF(IFLAG .GT. 0)GOTO 1002
            CALL PRSUB(N,28,ERRNOR,X,RHSTOT)
 1002       CONTINUE
C           NEXT, EVALUATE THE NEW CORRECTIONS S AND YPC FOR THE NON-LINEAR
C           EQUATIONS THAT MUST BE SOLVED AT THE NEW LEVEL K AFTER AN
C           INTERPOLATION FOR THE VALUES ACCEPTED AT LEVEL K-1.
C           WHEN NEWN IS TRUE , DFY IS EVALUATED DURING THE NEWTON
C           ITERATIONS .
            NEWN=.FALSE.
            ITLIM=3
            ITLIM2=5
            CALL SPLDCG(K,N,FU,S,IERROR)
            CALL DYDCG(K,N,H,FU,YPC,IERROR)
            DO 17 I=1,NP1
   17       YPCOLD(I)=YPC(I)
            DO 18 I=1,K
   18       ERRSAV(I)=ERRSAV(I)*16.**(-I)
            GOTO 38
   19       IF(YINT)GOTO 32
            DO 20 I=1,NP3
   20       RHSTOT(I)=0.0
            IF((BV0 .NE. 0.) .OR. (BVN .NE. 0.))GOTO 38
            CALL LINST(G0,ALF0,GN,ALFN,A,B,N,H,RHSTOT)
            GOTO 38
   32       CALL GETST(W,RHSTOT,WK,N,H)
            IF(IFLAG .EQ. 0)CALL PRSUB(N,28,TWOPI,X,RHSTOT)
C                MAIN PORTION OF NEWTON'S METHOD
C           WE ENTER HERE WITH AN INITIAL GUESS FOR Y, EITHER THE LINEAR
C           INTERPOLATION OF THE BOUNDARY DATA OR AN IMPROVED ESTIMATE
C           FROM A LOWER LEVEL AT THIS N OR A HIGHER LEVEL AT A SMALLER N
C           OR A USER SUPPLIED INITIAL GUESS.
C           AT LEVEL K=0 WE ALLOW 11 ITERATIONS BEFORE TERMINATING THE
C           PROCEDURE. AT HIGHER LEVELS, ONLY 4 ITERATIONS ARE ALLOWED.
C           AT LEVEL K=0 NEWN IS TRUE AND WE EVALUATE DFY AT EACH
C           ITERATE.  FOLLOWING AN INTERPOLATION ACCEPTED AT HIGH ORDER F,DFY,
C           AND DFYP ARE EVALUATED BEFORE ENTERING THE LOOP AND ARE NOT
C           REEVALUATED ON ENTRY.  DFY AND DFYP ARE NEVER EVALUATED
C           WITHIN THE LOOP IN THIS CASE.
C           NORMAL EXIT OF THE NEWTON ITERATION OCCURS WHEN THE NORM
C           (RNORM) OF THE CHANGE IN THE SOLUTION Y IS LESS THAN OR
C           EQUAL TO HALF THE USER'S TOLERANCE TIMES THE NORM OF Y.
C           ABNORMAL EXITS:
C             (1) THE NORM OF THE RESIDUAL INCREASES FROM STEP N TO
C                 N + 1, WHERE N > 0.
C                 OR
C             (2) THE NUMBER OF ITERATIONS EXCEEDS ITLIM AND CONVERGENCE
C                 IS NOT EXPECTED BY ITLIM2.
C           IF THE CHANGE IN Y (RNORM) INCREASES BUT THE RESIDUAL DECREASES,
```

```
C         THIS IS TAKEN AS A SIGN THAT ROUNDING ERROR IS PREVENTING
C         CONVERGENCE TO THE TOLERANCE AND THE PROGRAM GIVES AN ERROR
C         MESSAGE AND TERMINATES.
   38        INDEX=0
             DIVNEW=.FALSE.
             NOEVAL=.TRUE.
             IF((N .EQ. N0) .OR. (K .GT. 0))NOEVAL=.FALSE.
   40           CALL GETVAL(N,H,RHSTOT,Y,YP,AY2P)
                CALL BC(ALF0,BV0,G0,ALFN,BVN,GN,H,N,RHS,RHSTOT)
                RESN=AMAX1(ABS(RHS(1)),ABS(RHS(NP3)))
                IF(NOEVAL)GOTO 65
                DO 60 I=1,NP1
                YPTEMP=YP(I)-YPC(I)
                FU(I)=F(X(I),Y(I),YPTEMP)
                IF( .NOT. NEWN)GOTO 60
                E0(I+1)=-DFY(X(I),Y(I),YPTEMP)
                E1(I+1)=-DFYP(X(I),Y(I),YPTEMP)
   60           CONTINUE
                KOUNT1=KOUNT1+NP1
                IF( .NOT. NEWN)GOTO 65
                KOUNT2=KOUNT2+NP1
   65           NOEVAL=.FALSE.
                DO 70 I=1,NP1
                     RHS(I+1)=S(I+1)+FU(I)-AY2P(I)
                     IF(RESN .LT. ABS(RHS(I+1)))RESN=ABS(RHS(I+1))
   70                CONTINUE
                IJOB=2
                IF(NEWN)IJOB=0
                CALL MATMAN(W,RHS,E0,E1,WK,N,H,IJOB)
                RNORM=0.
                RTNORM=0.
                DO 100 I=1,NP3
                RHSTOT(I)=RHSTOT(I)+RHS(I)
                IF(RNORM .LT. ABS(RHS(I)))RNORM=ABS(RHS(I))
  100           IF(RTNORM .LT. ABS(RHSTOT(I)))RTNORM=ABS(RHSTOT(I))
C               AYNORM IS AN APPROXIMATION TO THE NORM OF Y BASED ON THE
C               THEORETICAL BOUND ON THE RATIO OF THE NORMS OF THE SPLINE COEF.
C               AND THE FUNCTION Y.
                AYNORM=RTNORM/3.
                INDEX=INDEX+1
                IF(INDEX .EQ. 1)RNORM1=RNORM
                EPS=EPSNWT*AYNORM
                IF(IFLAG .EQ. 0)PRINT 470,INDEX,RESN,RNORM
                IF(RNORM .LE. EPS)GOTO 390
                IF(INDEX .LE. 2) GOTO 120
                IF(RESN .GT. RSNSAV)GOTO 300
                IF((RNORM .GT. RNSAV) .AND. CONVOB)GOTO 350
  120           RNS2=RNSAV
                RNSAV=RNORM
                RSNSAV=RESN
                IF(INDEX .LE. ITLIM)GOTO 40
                IF(INDEX .GT. ITLIM2)GOTO 300
                IF((RNORM *(RNORM/RNS2)**(ITLIM2-INDEX)) .LT. EPS)GOTO 40
C            NEWTON+S METHOD IS DIVERGING AND WE MUST DOUBLE N IF
C            POSSIBLE.  IF THE FINAL NEWTON ITERATE SOLVES THE NON-
C            LINEAR SYSTEM BETTER THAN THE FIRST ITERATE, THEN WE GET
C            AN ERROR ESTIMATE FOR USE IN TESTING ORDER AND WE USE
```

```
C            VALUES INTERPOLATED FROM THE LAST ITERATE IN ORDER TO
C            START NEWTON+S METHOD AT LEVEL ZERO AND WITH N DOUBLED:
C            IF THE FINAL ITERATE WAS WORSE, HOWEVER, WE START WITH
C            THE LINEAR INTERPOLATION OF THE BOUNDARY DATA.
  300        DIVNEW=.TRUE.
             IF((RNORM1 .GT. RNORM) .OR. CONVOB)GOTO 400
             IF(N .GT. NMAXH)GOTO 800
             IF(YINT)GOTO 820
             N=2*N
             GOTO 2
C            WE NOW THINK THAT ROUNDING ERROR HAS CONTAMINATED OUR
C            RESULTS, SO WE PREPARE TO QUIT COMPLETELY AFTER ESTIMATING
C            THE ERROR WE DID ACHIEVE.
  350        RERROR=.TRUE.
             GOTO 400
C            WE NOTE FOR FUTURE REFERENCE THAT CONVERGENCE HAS BEEN
C            OBTAINED AT LEAST ONCE.
  390        CONVOB=.TRUE.
C            WE WANT TO ESTIMATE THE DISCRETIZATION ERROR, ESSENTIALLY
C            BY ONE STEP OF NEWTON+S METHOD.  WE SET UP THE RIGHT HAND
C            SIDE RHS NEEDED FOR THE LINEAR EQUATIONS OF THIS ONE STEP:
C           WHILE COMPUTING RHS WE WILL ALSO COMPUTE S AND YP, THE
C           CORRECTION (ESTIMATING LOCAL TRUNCATION ERROR) THAT WE WILL
C            NEED FOR THE NON-LINEAR EQUATIONS AT THE NEXT LEVEL UP IF
C            WE GET THAT FAR.
  400        K=K+1
             ITLIM=3
             ITLIM2=5
             CALL SPLDCG(K,N,FU,RHS,IERROR)
             CALL DYDCG(K,N,H,FU,YPC,IERROR)
             CALL BC(ALF0,BV0,G0,ALFN,BVN,GN,H,N,RHS,RHSTOT)
             DO 450 I=2,NP2
             STE=RHS(I)
             RHS(I)=S(I)-STE
             S(I)=STE
             RHS(I)=RHS(I)-E1(I)*(YPC(I-1)-YPCOLD(I-1))
  450        YPCOLD(I-1)=YPC(I-1)
C            WITH THE NEW RIGHT HAND SIDE, WE GENERATE AND SOLVE THE
C            NEW SYSTEM.
             CALL MATMAN(W,RHS,E0,E1,WK,N,H,2)
C
C            ERROR CONTROL AND DECISION CENTER
C            RHS NOW CONTAINS OUR ESTIMATE OF DISCRETIZATION ERROR.
C            WE COMPUTE ITS NORM IN ERRNOR, ALLOWING US TO TEST FOR
C            CONVERGENCE OF THE OVERALL ALGORITHM.
             NEWN=.FALSE.
             ERRNOR=0.
             YNORM=0.
             DO 460 I=1,NP1
             TE=ABS((RHS(I)+4.*RHS(I+1)+RHS(I+2))/6.)
             IF(TE .GT. ERRNOR)ERRNOR=TE
  460        IF(YNORM .LT. ABS(Y(I)))YNORM=ABS(Y(I))
             IF(IFLAG .GT. 0)GOTO 1001
             K1=K-1
             PRINT 470,INDEX,RESN,RNORM
             CALL PRSUB(N,K1,ERRNOR,X,RHSTOT)
 1001        CONTINUE
```

```
C             NEXT WE TEST OUR ESTIMATE OF OVERALL ERROR AGAINST
C             THE USER'S TOLERANCE.  ERRNOR ESTIMATES THE
C             DISCRETIZATION ERROR FOR THE EXACT SOLUTION OF THE
C             NON-LINEAR SYSTEM, WHILE RNORM ESTIMATES THE ERROR IN
C             SOLVING THAT NON-LINEAR SYSTEM.  ERREST ESTIMATES THE
C             TOTAL ERROR UNDER THE PESSIMISTIC ASSUMPTION THAT
C             THE OTHER ERRORS SUM.
              ERREST=ERRNOR+RNORM
              TOLLOC=TOL*YNORM
C             TEST FOR SUCCESS
              IF(ERREST .LT. TOLLOC)GOTO 900
C             TERMINATE ALGORITHM IF RERROR TRUE
              IF(RERROR)GOTO 980
C             IF K .LE.  MAXK (SO THAT THERE IS HISTORY TO COMPARE
C             AGAINST) WE NEXT CHECK WHETHER THE ERROR IS BEHAVING
C             AT THE CORRECT ORDER.
              IF(K .GT. MAXK)GOTO 480
              RAT=ERRSAV(K)/ERRNOR
              ERRSAV(K)=ERRNOR
              IF(IFLAG .EQ. 0)PRINT 475,RAT
C             THIS IS THE TEST FOR CORRECT ORDER.
              IF(RAT .LT. TEST1(K) .OR. RAT .GT. TEST2(K))GOTO 700
  480         ERRSAV(K)=ERRNOR
C             THE NEXT STATEMENT TESTS WHETHER THE PREVIOUS DEFERRED
C             CORRECTION IMPROVED THE ACCURACY ACCEPTABLY AND
C             WHETHER AN ERROR ESTIMATE WOULD BE ALLOWED AFTER
C             THE NEXT CORRECTION.
              IF(ERRNOR .GE. .1*ERROLD .OR. K+1 .GT. KMAX)GOTO 700
C             CHECK FOR NEWTON DIVERGENCE
              IF(DIVNEW)GOTO 700
              ERROLD=ERRNOR
C             GO AHEAD AND SOLVE THE SYSTEM AT THE HIGHER ORDER
              GOTO 38
C          WE NEED TO INCREASE N AND THE NEXT SECTION OF CODE HANDLES
C          THAT.  WE FIRST COMPUTE THE LEVEL INTERPOLATED VALUES WILL BE
C          ACCEPTED AT. THEN WE DO THE INTERPOLATION.
  700      IF(N .GT. NMAXH)GOTO 800
           MAXK=K
           NOLD=N
           N=2*N
           KINT=K
           ERROLD=AMIN1(ERROLD,ERRNOR)
C          DECIDE LEVEL TO INTERPOLATE TO
           DO 710 I=1,K
           IF(ERROLD .LT. ERRSAV(I)*16.**(-I))GOTO  710
           GOTO 720
  710      CONTINUE
  720      K=I-1
           IF(DIVNEW)K=0
C          INSPL PERFORMS THE INTERPOLATION AND FILLS IN Y VALUES.
           CALL INSPL(KINT,N,Y,W,S,RHSTOT,WK,K,H)
           GOTO 3
C       THIS IS THE N TOO BIG EXIT
  800 PRINT 815
        GOTO 900
C       820 IS THE ERROR EXIT TAKEN WHEN AN INITIAL GUESS FAILS.
  820 PRINT 825
```

```
C     THE FOLLOWING CODE SEGMENT PREPARES TO RETURN TO THE USER.
C     IN THE ABSENCE OF ERROR MESSAGES, THE RETURN IS SUCCESSFUL.
  900 TOL=ERREST
      IF(IFLAG .EQ. 0)PRINT 920,YNORM
      CALL SPLGEN(W,N,H,RHSTOT)
      RETURN
C     980 IS THE ROUNDING ERROR EXIT.
  980 PRINT 990
      GOTO 900
  470 FORMAT(I10,2E20.10)
  475 FORMAT(80X,F12.4)
  815 FORMAT(* N TOO BIG*)
  825 FORMAT(* ILLEGAL TO REINITIALIZE WHEN INITIAL GUESS GIVEN*)
  920 FORMAT(* YNORM = *,E20.10)
  990 FORMAT(*  ROUNDING ERROR EXIT*)
      END




      SUBROUTINE GETVAL(N,H,RHS,Y,YP,AY2P)
C     THIS ROUTINE TAKES THE VALUE OF THE SPLINE COEFFICIENTS
C     AND COMPUTES THE VALUES OF Y,YP, AND AY2P FOR USE
C     IN EVALUATING THE USER FUNCTIONS F,DFY,AND DFYP.
      DIMENSION RHS(1),Y(1),YP(1),AY2P(1)
      NP1=N+1
      DO 30 I=1,NP1
      Y(I)=(RHS(I)+4.*RHS(I+1)+RHS(I+2))/6.
   30 YP(I)=(RHS(I+2)-RHS(I))/(H*2.)
      AY2P(1)=7.*RHS(1)-16.5*RHS(2)+14.*RHS(3)-7.*RHS(4)
     1 +3.*RHS(5)-.5*RHS(6)
      DO 40 I=2,N
      AY2P(I)=.5*RHS(I-1)+4.*RHS(I)-9.*RHS(I+1)+4.*RHS(I+2)
     1 +.5*RHS(I+3)
   40 CONTINUE
      AY2P(NP1)=-.5*RHS(N-2)+3.*RHS(N-1)-7.*RHS(N)+14.*RHS(NP1)
     1 -16.5*RHS(N+2)+7.*RHS(N+3)
      DO 50 I=1,NP1
   50 AY2P(I)=AY2P(I)/(6.*H*H)
      RETURN
      END




      SUBROUTINE MATMAN(W,RHS,E0,E1,WK,N,H,IJOB)
C     THIS ROUTINE GENERATES THE SOLUTION MATRIX W FROM THE
C     VECTORS OF PARTIAL DERIVATIVE VALUES E0 AND E1.
C     DIV1 AND DIV2 ARE ASSUMED PRESERVED BETWEEN CALLS TO
C     FACILITATE REPEATED CALLS WITH THE SAME E0 AND E1
C     VALUES FOR DIFFERENT RIGHT HAND SIDES.  IJOB IS
C     DEFINED AS IN LEQT1B.
C     AFTER THE MATRIX IS GENERATED THE LINEAR SYSTEM
```

```
C     IS SOLVED BY CALLING LEQT1B.
      DIMENSION W(259,5),RHS(259),E0(259),E1(259),WK(777)
      HSQ=H*H
      NP1=N+1
      NP2=N+2
      NP3=N+3
      HSQ6I=1./(6.*H*H)
      IF(IJOB .EQ. 2)GOTO 100
C     ROWS 1 AND N+3 DEPEND ONLY ON THE BOUNDARY DATA WHICH WE STORE
C     IN E0 AND E1.
      W(1,3)=(E0(1)*H-3.*E1(1))/(6.*H)
      W(1,4)=2.*E0(1)/3.
      W(1,5)=(E0(1)*H+3.*E1(1))/(6.*H)
      W(NP3,1)=(E0(NP3)*H-3.*E1(NP3))/(6.*H)
      W(NP3,2)=2.*E0(NP3)/3.
      W(NP3,3)=(E0(NP3)*H+3.*E1(NP3))/(6.*H)
C     ROWS 3 THROUGH N+1 ARE STANDARD FIVE BAND ROWS BASED ON THE
C     DIFFERENTIAL EQUATION.
      DO 50 I=3,NP1
         CON1=3.*H*E1(I)
         CON2=HSQ*E0(I)
         W(I,2)=(4.-CON1+CON2)*HSQ6I
         W(I,3)=(-9.+4.*CON2)*HSQ6I
         W(I,4)=(4.+CON1+CON2)*HSQ6I
         W(I,1)=0.5*HSQ6I
   50    W(I,5)=0.5*HSQ6I
C     ROWS 2 AND N+2 HAVE SIX ELEMENTS IN THE ORIGINAL MATRIX; PRE-
C     ELIMINATION IS USED TO ELIMINATE TWO ELEMENTS OF EACH.
      CON1=3.*E1(2)*H
      CON2=HSQ*E0(2)
      DIV2=-2.0*(3.+W(4,4)/HSQ6I)
      W(2,2)=(7.-CON1+CON2)*HSQ6I+DIV2*W(3,1)
      W(2,3)=(-16.5+4.*CON2)*HSQ6I+DIV2*W(3,2)+W(4,1)
      W(2,4)=(14.+CON1+CON2)*HSQ6I+DIV2*W(3,3)+W(4,2)
      W(2,5)=-7.*HSQ6I+DIV2*W(3,4)+W(4,3)
      CON1=3.*E1(NP2)*H
      CON2=HSQ*E0(NP2)
      DIVN=-2.0*(3.+W(N,2)/HSQ6I)
      W(NP2,1)=-7.*HSQ6I+DIVN*W(NP1,2)+W(N,3)
      W(NP2,2)=(14.-CON1+CON2)*HSQ6I+DIVN*W(NP1,3)+W(N,4)
      W(NP2,3)=(-16.5+4.*CON2)*HSQ6I+DIVN*W(NP1,4)+W(N,5)
      W(NP2,4)=(7.+CON1+CON2)*HSQ6I+DIVN*W(NP1,5)
  100 CONTINUE
      IF(IJOB .EQ. 1)GOTO 150
      RHS(2)=RHS(2)+RHS(4)+DIV2*RHS(3)
      RHS(NP2)=RHS(NP2)+RHS(N)+DIVN*RHS(NP1)
  150 CALL LEQT1B(W,NP3,2,2,RHS,WK,IJOB,IER)
      IF(IER .NE. 0)STOP44
      RETURN
      END
```

```
      SUBROUTINE INSPL(K,N,Y,W,S,RHS,WK,KNEW,H)
C     THE DIFFERENCE CORRECTION GENERATION PART OF THE NEXT
C     THREE ROUTINES IS BASED ON U2DCG OF PEREYRA.
C     SEE =HIGH ORDER FINITE DIFFERENCE SOLUTION OF
C     DIFFERENTIAL EQUATIONS=, BY V. PEREYRA, STANFORD
C     UNIV. CS REPORT STAN-CS-73-348,APRIL,1973.
C     THIS ROUTINE DOES THE INTERPOLATION OF THE Y VALUES
C     AND THEN CALLS GETCO TO GET SPLINE COEFFICIENTS.
      DIMENSION Y(1),W(1),S(1),RHS(1),WK(1)
      DIMENSION A(28),C(28),CCON(28),AEXP(28)
      DATA CCON/2*0.,2.,0.,-2.,0.,2.,0.,11.3333333333333,0.,-198.,0.,
     1 2157.,0.,-12131.3333333333,0.,-3.48228666666666E5,0.,
     2 19492202.,0.,-5.94299148666666E8,0.,7.76269413533333E9,0.,
     3 6.07798554731333E11,0.,-6.84221476827757E13,0./
      DATA A/1.0,23*0.0/
      NO2=N/2
      NO21=NO2+1
      DO 10 I=1,NO21
   10 W(I)=Y(I)
      KK1=4*K
      II=NO21-KK1
      KK=KK1-1
      KMID=(KK1+1)/2
      KMID1=KMID-1
      KK1P1=KK1+1
      KKIIP1=KK+II+1
C            ASYMMETRIC APPROXIMATIONS NEAR LEFT BOUNDARY
      DO 20 I=1,KMID1
            XI=I+0.5
            CALL COEGEN(KK1,XI,C,A)
C        USE REFLECTION NEAR RIGHT BOUNDARY
            ACUM=0.
            DO 12 J=1,KK1
   12       ACUM=ACUM+C(KK1P1-J)*Y(II+J)
            IF(K.NE.0) S(KKIIP1-I)=ACUM
            IF(K.EQ.0) S(NO21+1)=ACUM
            ACUM=0
            DO 15 J=1,KK1
   15       ACUM=ACUM+C(J)*Y(J)
   20       S(I)=ACUM
C        SYMMETRIC APPROXIMATIONS CONSTANT THROUGHOUT CENTER RANGE
      XI=KMID+0.5
      CALL COEGEN(KK1,XI,C,A)
      NF=NO21-KMID
      DO 40 I=KMID,NF
            ACUM=0.
            II=I-KMID
            DO 38 J=1,KK1
   38       ACUM=ACUM+C(J)*Y(II+J)
   40       S(I)=ACUM
      DO 60 I=1,NO2
      I2=2*I
      Y(I2-1)=W(I)
   60 Y(I2)=S(I)
      Y(N+1)=W(NO21)
      NP1=N+1
```

```
      DO 70 I=1.NP1
   70 RHS(I+1)=Y(I)
      CALL GETCO(W,RHS,WK,N,H,KNEW)
      RETURN
      END





      SUBROUTINE SPLDCG(K,N,Y,S,IERROR)
C     THIS ROUTINE COMPUTES THE DIFFERENCE CORRECTION FOR
C     THE BASIC METHOD.   IT COMPUTES THE CORRECTION FOR
C     POINTS 1,...,N+1.   SEE THE REFERENCES ON DIFFERENCE
C     CORRECTION FOR THIS METHOD.
      DIMENSION A(28),Y(1),S(1),C(28),GA(28),GB(28)
      DATA  A/4*0.,6.66666666666667E-2,0.,1.19047619047619E-1,0.,
     1 -1.08888888888888,0.,8.18181818181818,0.,-33.3223443223443,0.,
     2 -725.472222222222,0.,-3185.00065359477,0.,-7.81972561403509E5,
     3 0.,8.40118413131313E6,0.,5.50542169142512E8,0.,
     4 -5.26324212944414E10,0.,2.66683028642333E12,0./
      DATA GA/4*0.,-1.93333333333333,-10.,-34.8809523809524,-105.,
     1 -285.755555555555,-714.,-1.75181818181818E3,-4620.,
     2 -1.31233223443223E4,-37466.,-9.77466388888888E4,186777.5,
     3 -2.14706660130718E5,-1.54422333333333E6,-1.47397205614035E7,
     4 -51816952.,7.52986007979798E7,1748890150.,6.90137631080917E9,
     5 -5.11509895563333E10,-6.86176242012174E11,-5.00936861343333E11,
     6 346536665345130.,1999784250626220./
      DATA GB/4*0.,-1.93333333333333,10.,-34.8809523809524,105.,
     1 -285.755555555555,714.,-1.75181818181818E3,4620.,
     2 -1.31233223443223E4,37466.,-9.77466388888888E4,-186777.5,
     3 -2.14706660130718E5,1.54422333333333E6,-1.47397205614035E7,
     4 51816952.,7.52986007979798E7,-1748890150.,6.90137631080917E9,
     5 5.11509895563333E10,-6.86176242012174E11,5.00936861343333E11,
     6 346536665345130.,-1999784250626220./
C                           CHECK INPUTS
      IF((K .LE. 0) .OR. (K .GT. (N-3)/4))GOTO 100
      KK1=4*(K+1)
      KK=KK1-1
      KMID=(KK1+1)/2
      IERROR=0
      KMID1=KMID-1
      KINT=KK1
      II=N-KK
      KK1P1=KK1+1
      KKIIP3=KK+II+3
C     SPECIAL APPROXIMATION AT LEFT BOUNDARY
      CALL COEGEN(KK1,1.,C,GA)
      ACUM=0.
      DO 2 J=1,KK1
    2 ACUM=ACUM+C(J)*Y(J)
      S(2)=ACUM
C            ASYMMETRIC APPROXIMATIONS NEAR LEFT BOUNDARY
      DO 5 I=2,KMID1
            XI=I
            CALL COEGEN(KK1,XI,C,A)
```

```
C           USE REFLECTION NEAR RIGHT BOUNDARY
          ACUM=0.
          DO 3 J=1,KK1
    3     ACUM=ACUM+C(KK1P1-J)*Y(II+J)
          S(KKIIP3-I)=ACUM
          ACUM=0
          DO 4 J=1,KK1
    4     ACUM=ACUM+C(J)*Y(J)
    5     S(I+1)=ACUM
C        SYMMETRIC APPROXIMATIONS CONSTANT THROUGHOUT CENTER RANGE
      KINT=KK
      XI=KMID
      CALL COEGEN(KINT,XI,C,A)
      NF=N+1-KINT+KMID
      DO 40 I=KMID,NF
          ACUM=0.
          II=I-KMID
          DO 38 J=1,KINT
   38     ACUM=ACUM+C(J)*Y(II+J)
   40     S(I+1)=ACUM
      XI=KK1
      II=N-KK
      CALL COEGEN(KK1,XI,C,GB)
      ACUM=0.
      DO 60 J=1,KK1
   60 ACUM=ACUM+C(J)*Y(II+J)
      S(N+2)=ACUM
      RETURN
  100 IERROR=1
      RETURN
      END



      SUBROUTINE DYDCG(K,N,H,Y,S,IERROR)
C     THIS ROUTINE COMPUTES THE CORRECTION TERM FOR YP IN THIS
C     METHOD.
      DIMENSION DCON(27),D(27),C(27),Y(1),S(1)
      DATA  DCON/3*0.,-3.33333333333333F-2,0.,7.93650793650794F-2,0.,
     1 -.194444444444444,0.,-9.09090909090909E-2,0.,12.6452991452991,
     2 0.,-226.777777777778,0.,2677.95506535948,0.,8184.68810916179,
     3 0.,-2.18828425555556E6,0.,1.02324117512077E8,0.,
     4 -2.80251085196407E9,0.,-2.75281377816667E10,0./
C                        CHECK INPUTS
      IF((K .LE. 0) .OR. (K .GT. (N-3)/4))GOTO 100
      KK1=4*(K+1)
      KK=KK1-1
      KMID=(KK1+1)/2
      IERROR=0
      KMID1=KMID-1
      II=N+1-KK
      KK1II=KK1+II
      DO 2 I=1,KK
    2     D(I)=H*DCON(I)
```

```
C           ASYMMETRIC APPROXIMATIONS AT OR NEAR LEFT BOUNDARY
      DO 5 I=1,KMID1
            XI=I
            CALL COEGEN(KK,XI,C,D)
C             USE REFLECTION NEAR RIGHT BOUNDARY
            ACUM=0.
            DO 3 J=1,KK
    3       ACUM=ACUM-C(KK1-J)*Y(II+J)
            S(KK1II-I)=ACUM
            ACUM=0
            DO 4 J=1,KK
    4       ACUM=ACUM+C(J)*Y(J)
    5       S(I)=ACUM
C        SYMMETRIC APPROXIMATIONS CONSTANT THROUGHOUT CENTER RANGE
      XI=KMID
      CALL COEGEN(KK,XI,C,D)
      NF=N+1-KK+KMID
      DO 40 I=KMID,NF
            ACUM=0.
            II=I-KMID
            DO 38 J=1,KK
   38       ACUM=ACUM+C(J)*Y(II+J)
   40       S(I)=ACUM
      RETURN
  100 IERROR=1
      RETURN
      END



      SUBROUTINE SPLGEN(W,N,H,RHS)
C     THIS ROUTINE TAKES THE SPLINE COEFFICIENTS USED
C     INTERNALLY AND COMPUTES THE RESULT GIVEN TO THE USER.
      DIMENSION W(259,5),RHS(259)
      NP1=N+1
      DO 30 I=1,NP1
      W(I,1)=(RHS(I)+4.*RHS(I+1)+RHS(I+2))/6.
      W(I,2)=(RHS(I+2)-RHS(I))/(2.*H)
      W(I,3)=(RHS(I)-2.*RHS(I+1)+RHS(I+2))/(H*H)
      IF(I .EQ. NP1)GOTO 40
   30 W(I,4)=(RHS(I+3)-RHS(I)+3.*(RHS(I+1)-RHS(I+2)))/(6.*H**3)
   40 RETURN
      END



      SUBROUTINE COEGEN(N,XNP,C,BB)
      DIMENSION C(1),BB(1)
C     ****                     *****                   *****
C     THIS IS A SLIGHTLY MODIFIED FORTRAN VERSION OF THE ALGOL
C     PROCEDURE PVAND, P.  901 OF
```

```
C       SOLUTION OF VANDERMONDE SYSTEMS OF EQUATIONS  BY
C      A. BJORCK AND V. PEREYRA, MATH. COMP. 24, PP.893-903 (1970)
C      WHERE A DESCRIPTION OF THE METHOD CAN BE FOUND.
C      ***      *****      ********       *****    *****
C      IN THIS IMPLEMENTATION, GIVEN GRID POINTS 1,....,N,
C      SEPARATED BY A CONSTANT H, THE COEFFICIENTS ARE COMPUTED FOR
C      A POINT (XNP- [XNP])*H BEYOND GRID POINT [XNP].
       DO 1 I=1,N
     1 C(I)=BB(I)
       NN=N-1
       DO 6 I=1,NN
            LL=N-I
            ALFI=I-XNP
            DO 6 J=1,LL
            K=N-J+1
     6      C(K)=C(K)-ALFI*C(K-1)
       DO 8 I=1,NN
            K=N-I
            XKIN=1./K
            KP1=K+1
            DO 8 J=KP1,N
                 C(J)=C(J)*XKIN
     8           C(J-1)=C(J-1)-C(J)
       RETURN
       END



       SUBROUTINE GETST(W,RHS,WK,N,H)
C      THIS ROUTINE IS USED WHEN INIT IS TRUE TO GET STARTED.
C      IT SIMPLY COPIES THE INITIAL GUESS AND CALLS
C      GETCO.
       DIMENSION X(1),RHS(1),W(259,5),WK(1)
       NP1=N+1
       DO 10 I=1,NP1
    10 RHS(I+1)=W(I,1)
       CALL GETCO(W,RHS,WK,N,H,0)
       RETURN
       END



       SUBROUTINE GETCO(W,RHS,WK,N,H,K)
C      THIS ROUTINE TAKES AN APPROXIMATION TO Y AT N+1 GRID POINTS,
C      GETS A NUMERICAL APPROXIMATION TO THE DERIVATIVE
C      EXPANSION FOR THE SPLINE WHICH THIS METHOD SEEKS AND THEN
C      CALLS LEQT1B TO SOLVE THE LINEAR EQUATIONS FOR THE SPLINE
C      COEFFICIENTS.
       DIMENSION X(1),RHS(1),W(259,5),WK(1),C(29),A(29),CTE(29)
       DATA C/2*0.,2.,0.,-2.,0.,2.,0.,11.3333333333333,0.,-198.,0.,
     1  2157.,0.,-12131.3333333333,0.,-3.48228666666666E5,0.,
```

```
     2  19492202.,0.,-5.94299148666666E8,0.,7.76269413533333E9,0.,
     3  6.07798554731333E11,0.,-6.84221476827757E13,0.,
     4  4.03224739307208E15/
      NP1=N+1
      NP3=N+3
      IF(K .GT. 6)STOP27
      KK=4*(K+1)
      DO 10 I=1,KK
   10 CTE(I)=C(I)/(H*H)
      CALL COEGEN(KK,1.,A,CTE)
      X1=0.
      X2=0.
      DO 20 I=1,KK
      X1=X1+A(I)*RHS(I+1)
   20 X2=X2+A(I)*RHS(NP3-I)
      RHS(1)=X1
      RHS(NP3)=X2
      W(1,3)=W(1,5)=1./(H*H)
      W(1,4)=-2./(H*H)
      W(N+3,1)=W(N+3,3)=1./(H*H)
      W(N+3,2)=-2./(H*H)
      NP2=N+2
      DO 100 I=2,NP2
      W(I,1)=W(I,5)=0.
      W(I,2)=W(I,4)=1./6.
  100 W(I,3)=2./3.
      CALL LEQT1B(W,N+3,2,2,RHS,WK,0,IER)
      IF(IER .NE. 0)STOP33
      RETURN
      END




      SUBROUTINE BC(ALF0,BV0,G0,ALFN,BVN,GN,H,N,RHS,RHSTOT)
C     THIS ROUTINE HANDLES THE BOUNDARY CONDITIONS FOR THE CASE WHEN
C     YP IS PRESENT IN THE BOUNDARY CONDITIONS.  IT COMPUTES THE
C     AMOUNT BY WHICH THE CURRENT SOLUTION FAILS TO SATISFY THE
C     BOUNDARY CONDITIONS AND RETURNS THOSE VALUES AS RHS(1) AND
C     RHS(N+3).  THIS ROUTINE WORKS DIRECTLY ON THE SPLINE
C     COEFFICIENTS AND DOES NOT TRY TO DO DIFFERENCE CORRECTION
C     ON YP AS IS REQUIRED BY THE THEORY.  IT WOULD PROBABLY BE
C     EASIER TO DO THAT DIRECTLY ON THE Y AND YP VALUES THAN ON THE
C     COEFFICIENTS.
      DIMENSION RHS(1),RHSTOT(1),BCO0(3),BCON(3)
      BCO0(1)=(ALF0*H-3.*BV0)/(6.*H)
      BCO0(2)=2.*ALF0/3.
      BCO0(3)=(ALF0*H+3.*BV0)/(6.*H)
      BCON(1)=(ALFN*H-3.*BVN)/(6.*H)
      BCON(2)=2.*ALFN/3.
      BCON(3)=(ALFN*H+3.*BVN)/(6.*H)
      TE1=0.
      TE2=0.
      DO 10 I=1,3
      TE1=TE1+BCO0(I)*RHSTOT(I)
```

```
      DATA NMAXH /128/
C     CHECK THE VALIDITY OF THE INPUT PARAMETERS
```



```
   10 TE2=TE2+RCON(I)*RHSTOT(N+I)
      RHS(1)=G0-TE1
      RHS(N+3)=GN-TE2
      RETURN
      END



      SUBROUTINE LINST(G0,ALF0,GN,ALFN,A,B,N,H,RHSTOT)
      DIMENSION RHSTOT(1)
C     THIS ROUTINE PRODUCES THE SPLINE COEFFICIENTS FOR A
C     LINEAR INTERPOLATION OF THE BOUNDARY DATA WHEN YP IS NOT
C     INVOLVED IN THE BOUNDARY CONDITIONS.
      Y1=G0/ALF0
      YN=GN/ALFN
      SLOP=(YN-Y1)/(B-A)
      RHSTOT(2)=Y1
      DO 30 I=1,N
   30 RHSTOT(I+2)=RHSTOT(2)+I*H*SLOP
      RHSTOT(1)=2.*RHSTOT(2)-RHSTOT(3)
      RHSTOT(N+3)=2.*RHSTOT(N+2)-RHSTOT(N+1)
      RETURN
      END



      SUBROUTINE LEQT1B (A,N,NLC,NUC,B,XL,IJOB,IER)
C
C     THIS ROUTINE IS ADAPTED FROM THE ROUTINE OF THE SAME NAME
C     IN THE IMSL LIBRARY.  THE CHANGES ARE:
C          (1) THE ROW DIMENSION OF A AND XL IS FIXED AT 259.
C          (2) B IS A VECTOR RATHER THAN A MATRIX.
C
C
C-LEQT1B--------S--------LIBRARY 3----------------------------------------
C
C   FUNCTION            - MATRIX DECOMPOSITION, LINEAR EQUATION
C                           SOLUTION - SPACE ECONOMIZER SOLUTION -
C                           BAND STORAGE MODE
C   USAGE               - CALL LEQT1B (A,N,NLC,NUC,B,IJOB,IER)
C   PARAMETERS   A      - INPUT/OUTPUT MATRIX OF DIMENSION N BY
C                           (NUC+NLC+1). SEE PARAMETER IJOB.
C                N      - ORDER OF MATRIX A AND THE NUMBER OF ROWS IN
C                           B. (INPUT)
C                NLC    - NUMBER OF LOWER CODIAGONALS IN MATRIX A.
C                           (INPUT)
C                NUC    - NUMBER OF UPPER CODIAGONALS IN MATRIX A.
C                           (INPUT)
C                IA     - ROW DIMENSION OF A AS SPECIFIED IN THE
C                           CALLING PROGRAM. (259)
C                B      - INPUT/OUTPUT MATRIX OF DIMENSION N BY M.
```

```
C         FILL IN Y VALUES AFTER DOUBLING N
          CALL GETVAL(N,H,RHSTOT,Y,YP,AY2P)
C              WE USE DIFFERENT YP VALUES AFTER AN INTERPOLATION
```



```
C                             ON INPUT, B CONTAINS THE M RIGHT-HAND SIDES
C                             OF THE EQUATION AX = B. ON OUTPUT, THE
C                             SOLUTION MATRIX X REPLACES B. IF IJOB = 1,
C                             B IS NOT USED.
C                             THIS VERSION ASSUMES ONE RIGHT HAND SIDE
C                  XL       - INPUT/OUTPUT MATRIX OF SIZE 777 WHICH CONTAINS
C                             PIVOTING INFORMATION WHICH SHOULD NOT BE
C                             DESTROYED BY THE CALLING PROGRAM.
C                  IB       - ROW DIMENSION OF B AS SPECIFIED IN THE
C                             CALLING PROGRAM. (259)
C                  IJOB     - INPUT OPTION PARAMETER. IJOB = I IMPLIES WHEN
C                               I = 0, FACTOR THE MATRIX A AND SOLVE THE
C                                 EQUATION AX = B. ON INPUT, A CONTAINS THE
C                                 COEFFICIENT MATRIX OF THE EQUATION AX = B,
C                                 WHERE A IS ASSUMED TO BE AN N BY N BAND
C                                 MATRIX. A IS STORED IN BAND STORAGE MODE
C                                 AND THEREFORE HAS DIMENSION N BY
C                                 (NLC+NUC+1). ON OUTPUT, A IS REPLACED
C                                 BY THE U MATRIX OF THE L-U DECOMPOSITION
C                                 OF A ROWWISE PERMUTATION OF MATRIX A. U IS
C                                 STORED IN BAND STORAGE MODE.
C                               I = 1, FACTOR THE MATRIX A. A CONTAINS THE
C                                 SAME INPUT/OUTPUT INFORMATION AS IF
C                                 IJOB = 0.
C                               I = 2, SOLVE THE EQUATION AX = B. THIS
C                                 OPTION IMPLIES THAT LEQT1B HAS ALREADY
C                                 BEEN CALLED USING IJOB = 0 OR 1 SO THAT
C                                 THE MATRIX A HAS ALREADY BEEN FACTORED.
C                                 IN THIS CASE, OUTPUT MATRIX A
C                                 MUST HAVE BEEN SAVED FOR REUSE IN THE
C                                 CALL TO LEQT1B.
C                  IER      - ERROR PARAMETER.
C                             TERMINAL ERROR = 128+N.
C                               N = 1 INDICATES THAT MATRIX A IS
C                                 ALGORITHMICALLY SINGULAR. (SEE THE
C                                 CHAPTER L PRELUDE).
      DIMENSION            A(259,1),XL(259,3),B(259)
      DATA                 ZERO/0./,ONE/1.0/
      IER = 0
      JBEG = NLC+1
      NLC1 = JBEG
      IF (IJOB .EQ. 2) GO TO 80
      RN = N
C                                      RESTRUCTURE THE MATRIX
C                                      FIND RECIPROCAL OF THE LARGEST
C                                      ABSOLUTE VALUE IN ROW I
      I = 1
      NC = JBEG+NUC
      NN = NC
      JEND = NC
      IF (N .EQ. 1 .OR. NLC .EQ. 0) GO TO 25
    5 K = 1
      P = ZERO
      DO 10 J = JBEG,JEND
         A(I,K) = A(I,J)
         Q =  ABS(A(I,K))
         IF (Q .GT. P) P = Q
```

```
C          (2) THE NUMBER OF ITERATIONS EXCEEDS ITLIM AND CONVERGENCE
C              IS NOT EXPECTED BY ITLIM2.
C          IF THE CHANGE IN Y (RNORM) INCREASES BUT THE RESIDUAL DECREASES.
```



```
      K = K+1
   10 CONTINUE
      IF (P .EQ. ZERO) GO TO 135
      XL(I,NLC1) = ONE/P
      IF (K .GT. NC) GO TO 20
      DO 15 J = K,NC
         A(I,J) = ZERO
   15 CONTINUE
   20 I = I+1
      JBEG = JBEG-1
      IF (JEND-JBEG .EQ. N) JEND = JEND-1
      IF (I .LE. NLC) GO TO 5
      JBEG = I
      NN = JEND
   25 JEND = N-NUC
      DO 40 I = JBEG,N
         P = ZERO
         DO 30 J = 1,NN
            Q =  ABS(A(I,J))
            IF (Q .GT. P) P = Q
   30    CONTINUE
         IF (P .EQ. ZERO) GO TO 135
         XL(I,NLC1) = ONE/P
         IF (I .EQ. JEND) GO TO 37
         IF (I .LT. JEND) GO TO 40
         K = NN+1
         DO 35 J = K,NC
            A(I,J) = ZERO
   35    CONTINUE
   37    NN = NN-1
   40 CONTINUE
      L = NLC
C                                 L-U DECOMPOSITION
      DO 75 K = 1,N
         P =  ABS(A(K,1))*XL(K,NLC1)
         I = K
         IF (L .LT. N) L = L+1
         K1 = K+1
         IF (K1 .GT. L) GO TO 50
         DO 45 J = K1,L
            Q = ABS(A(J,1))*XL(J,NLC1)
            IF (Q .LE. P) GO TO 45
            P = Q
            I = J
   45    CONTINUE
   50    XL(I,NLC1) = XL(K,NLC1)
         XL(K,NLC1) = I
C                                 SINGULARITY FOUND
         IF (RN+P .EQ. RN) GO TO 135
C                                 INTERCHANGE ROWS I AND K
         IF (K .EQ. I) GO TO 60
         DO 55 J = 1,NC
            P = A(K,J)
            A(K,J) = A(I,J)
            A(I,J) = P
   55    CONTINUE
   60    IF (K1 .GT. L) GO TO 75
```

```
C           POSSIBLE.  IF THE FINAL NEWTON ITERATE SOLVES THE NON-
C           LINEAR SYSTEM BETTER THAN THE FIRST ITERATE, THEN WE GET
C           AN ERROR ESTIMATE FOR USE IN TESTING ORDER AND WE USE
```



```
         DO 70 I = K1,L
            P = A(I,1)/A(K,1)
            IK = I-K
            XL(K1,IK) = P
            DO 65 J = 2,NC
               A(I,J-1) = A(I,J)-P*A(K,J)
   65       CONTINUE
            A(I,NC) = ZERO
   70    CONTINUE
   75 CONTINUE
      IF (IJOB .EQ. 1) GO TO 9005
C                                                 FORWARD SUBSTITUTION
   80 L = NLC
      DO 105 K = 1,N
         I = XL(K,NLC1)
         IF (I .EQ. K) GO TO 90
            P = B(K)
            B(K) = B(I)
            B(I) = P
   90    IF (L .LT. N) L = L+1
         K1 = K+1
         IF (K1 .GT. L) GO TO 105
         DO 100 I = K1,L
            IK = I-K
            P = XL(K1,IK)
               B(I) = B(I)-P*B(K)
  100    CONTINUE
  105 CONTINUE
C                                                 BACKWARD SUBSTITUTION
      JBEG = NUC+NLC
         L = 1
         K1 = N+1
         DO 120 I = 1,N
            K = K1-I
            P = B(K)
            IF (L .EQ. 1) GO TO 115
            DO 110 KK = 2,L
               IK = KK+K
               P = P-A(K,KK)*B(IK-1)
  110       CONTINUE
  115       B(K) = P/A(K,1)
            IF (L .LE. JBEG) L = L+1
  120    CONTINUE
      GO TO 9005
  135 IER = 129
 9000 CONTINUE
 9005 RETURN
      END
```

```
      PRINT 470,INDEX,RESN,RNORM
      CALL PRSUB(N,K1,ERRNOR,X,RHSTOT)
1001  CONTINUE
```

Unclassified

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) University of Texas at Austin	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP --

3. REPORT TITLE

6 Extrapolation with spline-collocation methods for two-point boundary-value problems II: C^2-cubics with detailed results

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Center for Numerical Analysis

5. AUTHOR(S) (First name, middle initial, last name)

10 James W./Daniel andrew J/Martin

6. REPORT DATE 11 August 1977 12 77p.	7a. TOTAL NO. OF PAGES 76	7b. NO. OF REFS 26
8a. CONTRACT OR GRANT NO. 15 N00014-76-C-0275 b. PROJECT NO. c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) 14 CNA-125 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES --	12. SPONSORING MILITARY ACTIVITY Mathematics Branch, Office of Naval Research, Washington D.C.

13. ABSTRACT

The methodology is very briefly described and then numerical results are presented for an implementation of a smooth-cubic-spline-collocation procedure accelerated via iterated deferred corrections to obtain approximate solutions, accurate to a prescribed tolerance, of two-point boundary-value problems for second-order scalar ordinary differential equations. The results are similar to those obtained via a less generally applicable finite-difference-oriented code described elsewhere which competes well with the best codes available.

Key Words: Boundary-value, collocation, spline, deferred correction

406262 LB

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6801

Unclassified

Security Classification